Measuring Self-efficacy in Secure Programming

Matt Bishop, Ida Ngambeki, Shiven Mian, Jun Dai, Phillip Nico

Measuring Self-Efficacy in Secure Programming

Matt Bishop[1][0000-0002-7301-7060], Ida Ngambeki[2][0000-0001-7191-2179], Shiven Mian[3][0000-0002-7730-5254], Jun Dai[4][0000-0002-6890-6429], Phillip Nico[5][0000-0002-7405-2546]

[1] University of California, Davis; *email:* mabishop@ucdavis.edu
[2] Purdue University; *email:* ingambek@purdue.edu
[3] University of California, Davis; *email:* smian@ucdavis.edu
[4] California State University, Sacramento; *email:* jun.dai@csus.edu
[5] California Polytechnic State University; *email:* pnico@calpoly.edu

Abstract. Computing students are not receiving enough education and practice in secure programming. A key part of being able to successfully implement secure programming practices is the development of secure programming self-efficacy. This paper examines the development of a scale to measure secure programming self-efficacy among students participating in a secure programming clinic (SPC). The results show that the secure programming self-efficacy scale is a reliable and useful measure that correlates satisfactorily with related measures of programming expertise. This measure can be used in secure programming courses and other learning environments to assess students' secure programming efficacy.

Keywords: Self Efficacy, Secure Programming, Secure Programming Clinic.

1 Introduction

Bugs in computer programs are as old as programming. Indeed, the term "bug" is said to have come from Grace Murray Hopper in 1946, when she investigated an error in the Mark II computer at the Harvard Computation Laboratory. A moth was trapped in the relay, causing the problem. This became the first computer bug. As computing became more widespread, concern about the security of information on the systems grew. The meaning of the term "security" was defined by a (formal or informal) statement, the security policy, which varied depending on the organization. Common to all definitions was the notion of escalating the privileges of a process so it could perform functions it was not supposed to. And a common way to do this was to exploit bugs in programs.

Programs were, and are, usually written non-robustly. The Common Vulnerabilities and Exposures (CVE) database, which enumerates software vulnerabilities, has over 152,000 entries [1]. Programming errors have been found in electronic voting systems [2] and automobiles [3]. The lack of robustness inspired Weinberg's Second Law, which says that if builders built buildings like programmers wrote programs, the first woodpecker to come along would destroy civilization [4].

Implicitly, programs trust the environment they execute in, and that the input they receive is well-formed. Robust programs do not do this; they examine those parts of the environment that affect how the program works to ensure it is as required. Input is examined to ensure it is well-formed. The code is written to handle failures by reporting the error conditions and either recovering or terminating gracefully. This type of programming is called "robust programming". The term "secure programming" is often used as a synonym to robust programming. Technically, there is a difference: a "secure program" is a robust program that satisfies a given security policy. However, many security problems arise from non-robust programming. So, the focus of improving programming is on improving the robustness, because it is reasonable to assume that it will eliminate many security vulnerabilities.

Most first programming classes teach robust programming. Students are taught to validate input, check that references to arrays and lists are within bounds, and so forth — all elements of good programming style. As they progress, the focus of classes shifts to the content of the class, such as data structures or networking, and programs are no longer graded based on their robustness. Because of this, students' knowledge and abilities to program robustly is not used and so grows rusty due to lack of practice. Worse, as many introductory courses move to higher-level, friendlier programming languages such as Python, students may not have even encountered whole classes of non-robust practices during this formative period and will be unprepared when they encounter less friendly languages like C. This is similar to writing. In many disciplines, essays and written answers to homework and test questions are sloppy; the questions are answered, but the writing often makes understanding the answer difficult or the writing is jumbled. English departments and law schools are well aware of this problem, and in response have developed "writing clinics". These clinics do not judge whether the writing answers the questions in the homework. Instead, they look at the structure and grammar of the essay and make suggestions on how to improve the writing.

This suggests that a "secure programming clinic" (SPC) performing analogous reviews of programs might help improve the quality of programs [5,6]. The SPC would review programs for robustness. It would not determine whether the program had met the requirements of the assignment. This way, students have numerous opportunities to develop good programming style and the practice of robustness throughout their educational career. A key benefit of the SPC is to serve as a place where students can find information about secure and robust programming as well as tools to help them check their code for vulnerabilities and other coding problems. They can make appointments with a clinician, who is typically a faculty member or an experienced graduate student, to review programs and can use the materials at the SPC to learn more or reinforce what they have learned. The SPC can take many forms. As described above, one form is that of the traditional "writing clinic". Another is to have the clinic review programs after they are turned in, and report problems in robustness to the graders; the final grade would take this into account. A variant is to allow students to fix the problems in robustness and resubmit the program to have the robustness part regraded. Other variants are possible. By appropriately requiring use of the SPC, classes and instructors can continue to emphasize the importance of robust, secure programming in a way that does not impact instruction time. This will continue throughout and, with hope, after the student's work at the academic institution.

One of the key goals of the SPC is to increase students' self-efficacy viz. their confidence in their knowledge of secure programming and their ability to complete secure programming tasks. Prior work has shown SPCs to be effective in developing students' expertise in Secure Programming concepts [7]. It was also found that the SPCs' overall efficacy was highly dependent on adjustment of the clinic structure to the contextual peculiarities of the sites where they were deployed [8]. These findings suggest that students' prior experience is related to the effectiveness of their clinic experience. We posit that this is due to a connection between students' experiences, the development of expertise, and students' self-efficacy. Proving any such correlation would require us to have quantitative measures for each of them, however, no reliable means to measure secure programming self-efficacy exists as of now.

This paper reports on efforts to develop a reliable measure of secure programming self-efficacy. In Section 2, we first discuss self-efficacy and its role in the development of expertise. Section 3 then reports on efforts to develop and validate a secure programming self-efficacy scale. Section 4 explores the validity of the scale by examining its relationship to secure programming knowledge and general programming experience, and Section 5 summarizes our contributions and avenues for future work.

2 Background and Related Work

In general, self-efficacy can be described as an individual's confidence in themselves, their confidence that the capabilities they possess are effective to accomplish a specific task or thrive in a certain situation. Formally, self-efficacy is defined as an individual's belief about his or her ability/capability to complete a specific task [9]. This theory was first postulated by Albert Bandura, a well-known social-cognitive psychologist in 1977 which was earlier added to his original Social Learning Theory (SLT) and revised into Social Cognitive Theory in 1986 [10]. According to Bandura, a person's efficacy beliefs are largely based on four sources:

- **Enactive Mastery Experiences**, i.e. actual performance of a task, or familiarity with a situation.
- **Vicarious Experience**, i.e. observation of others performing a task and succeeding.
- **Verbal Persuasions**, i.e. encouragement or discouragement from peers, verbal or otherwise which aid individuals to overcome self-doubt.
- **Physiological and Affective States**, i.e. relating to body or physical states as opposed to mind or psychological states.

Among these four, Enactive Mastery Experience is the most significant source of Self-Efficacy [11]. This source is strongly related to work on the development of expertise as students develop from novices to advanced beginners, to competent, to proficient, to expert. Students progress through these stages as a result of the accumulation of knowledge, time spent immersed in the subject, and repeated practice and application of knowledge in different contexts.

How does Self-Efficacy impact learning and expertise? While the domain-specific knowledge and intellectual abilities of a student play a great role in academic success, self-efficacy is another significant characteristic that should not be overlooked. Several studies based on this theory by various researchers have demonstrated that students with higher self-efficacy are more successful academically, as they are self-regulated and believe in their own abilities. According to Bandura's theory, the self-efficacy of an individual influences 1) the amount of effort expended, 2) the type of coping strategies adopted, 3) the cognitive strategies used while solving problems, 4) persistence at the time of failure, and 5) their performance outcomes [12]. In learning conditions, especially in programming, these attributes play a vital role. Since programming is a highly cognitive activity, students come across difficulties, problem solving, failure and complicated situations frequently. In a study aimed at exploring factors affecting a pre-service computer science teacher's 'attitude towards computer programming' (ATCP), one of the factors examined was computer programming self-efficacy. A computer programming self-efficacy scale was used to collect computer programming self-efficacy data. Gurer et al. [13] found that "there was a positive and significant correlation (r = 0.738, p < 0.01) between students' computer programming self-efficacy and their ATCP . Moreover, it was found that computer programming self-efficacy was a significant variable in predicting ATCP" [13].

How is Self-Efficacy measured? Multiple self-efficacy scales exist in literature that either measure generalized self-efficacy or measure efficacy belief specific to domains like reading/writing, mathematics, and using computer software [14]. The Self-Efficacy Scale by Sherer and Adams [15] was developed to assess expectancies of self-efficacy. This Self-Efficacy Scale has two subscales, both with adequate reliability, the General Self-Efficacy sub scale (Cronbach α = 0.86) and the Social Self-Efficacy subscale (Cronbach α = 0.71). The "general self-efficacy subscale predicted past success in vocational, educational, and military areas. The social self-efficacy subscale predicted past vocational success" [15]. There are examples of self-efficacy scales adapted specifically to programming. For example, Ramalingam and Wiedenbeck [14] developed and established the Computer Programming Self-Efficacy Scale, based on the three dimensions (magnitude, strength and generality) of self-efficacy in Bandura's theory. The scale involves answering thirty-three items in ten minutes. These items ask students to judge their competence in various programming tasks in object-oriented C++, to make the scale domain specific. Moreover, the items were reviewed by self-efficacy theory and C++ experts. The results showed that the reliability of these scores was 0.97 [14]. However, in order to keep the scale short and make it applicable specifically to secure programming, we adapted the general self-efficacy scale by Chen et al. [16] to measure secure programming self-efficacy (see Section 3).

What are the implications for secure programming? As Bandura stresses, mastery experiences are the most effective source of self-efficacy [11]. This can be applied to the development of secure programming self-efficacy. More practice increases students' confidence in their ability to write secure programs or learn robust coding practices. Both students' self-efficacy and performance are shaped by their prior experience and expertise before they come to the SPC, and students come to the SPC from a wide variety of backgrounds ranging from introductory programming students to seniors studying operating systems or computer security.

Students develop expertise and efficacy in a variety of ways, which include classroom activities and programming projects. In addition to these, though, many students participate in extracurricular activities that could boost the development of expertise and efficacy. Such activities include programming competitions, participation in programming message boards, technical internships, and online gaming. We developed a way to measure secure programming self-efficacy as described below in Section 3.

3 Study 1: Developing and Validating the Secure Programming Self-Efficacy Scale

In order to measure secure programming self-efficacy, we developed a secure programming self-efficacy scale. We based the scale on the General Self-Efficacy (GSE) Scale developed by Chen et. al [16]. We selected this scale because it is widely cited, strongly validated, and flexible. Since general programming knowledge is an important element in secure programming, we included a programming sub-scale. We constructed questions about programming self-efficacy and secure programming self-efficacy in the same form as those on the GSE Scale. Each scale consisted of eight items. A team of five secure programming and cybersecurity education faculty who teach secure programming then examined the scale items to ensure that they were clearly worded, consistent with the GSE definitions, consistent with self-efficacy theory and displayed no redundancy. The experts also assured the logical validity of the scales ensuring that it measured elements of programming self-efficacy and secure programming self-efficacy.

Sample and Procedure

Participants in the scale validation were 101 undergraduates (21% female) enrolled in a computer science and computer engineering majors at a large lower-Pacific university. The participants were 2nd to 5th year students enrolled in a secure programming course (2nd yr. = 14%, 3rd yr. = 21%, 4th yr. = 49%, 5th yr. = 15%; the remaining 1% is due to rounding). Participants completed a survey containing the self-efficacy scales towards the end of the semester. This scale asked students to rank the extent to which the student agreed or disagreed with the statements on a 1-5 scale. For example, if a student indicated "5" to the first statement, that means they are "very" confident in their ability to program; in contrast, if a "1" were given, that means they are "not" confident in his/her ability to program. In order to control order effects, the order in which the items appeared in the survey was randomized.

Results and Discussion

The results of the survey indicated that the proposed secure programming scale performed well. An analysis of the internal consistency of the scale yielded a Cronbach's alpha of 0.86. A principle components analysis yielded two factors for the 16 items (Table 1). The two factors loaded exactly along the programming and secure programming items on the scale. Each factor or subscale had a high internal consistency ($\alpha = 0.91$, $\alpha = 0.78$).

The results of the survey demonstrated that students generally scored higher on the programming self-efficacy scale (M = 3.95) than they did on secure programming self-efficacy (M = 3.08). This suggests that students are more confident in their ability to program than they are in their ability to program securely. This could be explained by the fact that most of these students have extensive programming expertise, having engaged in programming since at least their first year of college if not even earlier. However, for many if not most of them, this was their first course in secure programming. Most computer science and computer engineering programs place a greater emphasis on functionality than security. Security is therefore approached as a separate topic in a different course taken later in the program rather than integrated into all programming courses.

Table 1. Secure programming self-efficacy scale and reliability

Item	M	SD
[1]In general I am confident in my ability to program	3.70	0.87
[1]Compared to other people, I can program fairly well	3.44	0.79
[1]I believe I am good at programming	3.61	0.93
[1]I am confident in my ability to solve programming problems	3.72	0.86
[1]I enjoy programming	4.14	0.90
[1]I like to understand how programs work	4.33	0.88
[1]I enjoy my computer science classes	4.30	0.73
[1]I am interested in designing new programs	4.32	0.85
[2]I am familiar with secure programming	2.62	0.85
[2]I think secure programming is important	4.23	0.95
[2]I think it is important that my programs are secure/robust	4.15	0.98
[2]I will be able to successfully complete assignments in this class	3.70	0.92
[2]I check my programs specifically for security flaws	2.54	0.93
[2]I am able to identify security issues in my programs	2.54	0.99
[2]I am confident that I can produce programs without major security flaws	2.48	0.97
[2]I am confident that I can recognize security flaws in others' programs	2.40	0.91

[1]programming self-efficacy; [2]secure programming self-efficacy

Overall α = 0.86; programming self-efficacy α = 0.91; secure programming self-efficacy α = 0.78; N = 101;

4 Study 2: Examining the predictive validity of the Secure Programming Self-Efficacy Scale

In Study 2, we examined the predictive validity of the secure programming self-efficacy scale. We conducted this study in order to examine the relationship of secure programming self-efficacy and programming efficacy to related variables. This allows us to make inferences about discriminant, convergent, and predictive validity.

Sample and Procedure

Participants in Study 2 were 65 students (13.8% female) at a large lower-Pacific university. This was a different university than that in Study 1. Participants were 4th yr. and 5th yr. students (4th yr. = 69%, 5th yr. = 31%). All students were computer science majors enrolled in a secure programming course. Participants completed an electronic survey towards the end of the course. The survey contained demographic questions, questions about student performance, questions about students' other activities related to programming and expertise. To relate students' understanding of secure coding to their expertise and confidence levels, students completed questions related to their prior experiences and expertise.

Students' performance was measured using a series of 45 conceptual questions about secure programming. The pool of questions was developed and validated by testing it at four US universities in a previous study [7]. The generation of the questions was based on a concept map that we built to epistemologically depict the important sets of secure programming objects [17]. The objects were classified into ten categories: Inputs, Assumptions, Bad Code, Programming Development Environment, Software Assurance Tools, Algorithms, Input Validation, Memory Management, Code Design, Authoritative Cryptography. Development of the concept map based on input from subject matter experts is detailed in a related paper [18]. The questions diagnosed students' conceptual understanding of secure programming using multiple-choice questions. The questions contained carefully crafted distractors and a single correct option. This was to ensure that the students can only get to the correct answer based on truly understanding the concept instead of eliminating the obviously wrongful options.

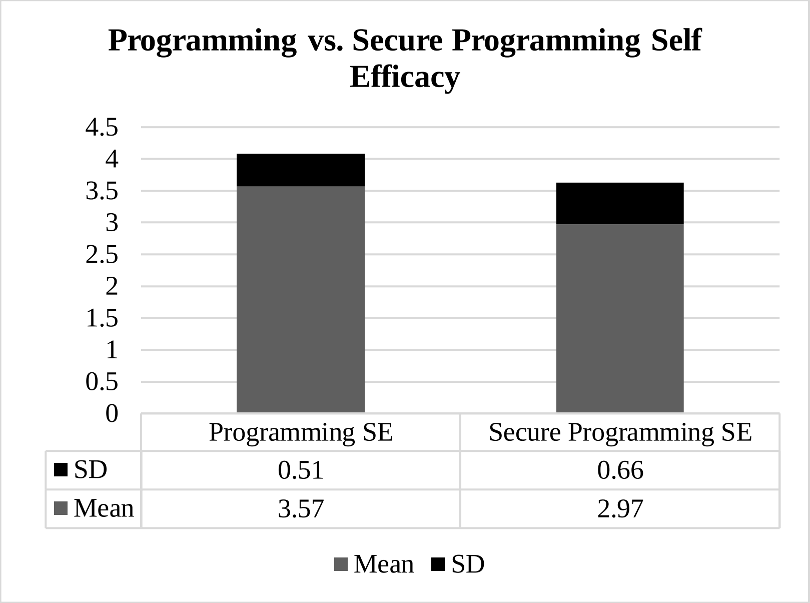

Fig. 1. Programming vs. secure programming self-efficacy

Results and Discussion

Students were measured on both general programming self-efficacy and secure programming self-efficacy. The expectation was that there would be a strong correlation between programming self-efficacy and secure programming self-efficacy. The study found that there was a significant correlation between secure programming self-efficacy and programming self-efficacy $r(65) = 0.86$, $p < 0.01$. Since secure programming is a subsidiary skill to programming, students would have to first develop confidence in their ability to program before they could be confident in their ability to program securely. Students expressed significantly less secure programming self-efficacy than general programming self-efficacy, i.e. students felt more capable about their ability to program than their knowledge of secure programming (Figure 1). A paired samples t-test showed that students programming self-efficacy (M = 3.57, SD = 0.51) is significantly higher than their secure programming self-efficacy (M = 2.97, SD = 0.66); $t(65) = 0.86$, $p < 0.001$.

We also examined the effect of gender on secure programming self-efficacy. We found that gender did not have a significant effect on student self-efficacy. Male and female students scored similarly on both programming (Male, M = 3.54, SD = 0.50; Female, M = 3.69, SD = 0.56) and secure programming (Male, M = 2.95, SD = 0.67; Female, M = 3.11, SD = 0.68) self-efficacy. An independent samples t-test comparing males and females found no significant differences by gender. However, this may be due to the number of women in the sample, as this group had only nine females representing only 13.8% of the population.

Students' knowledge of secure programming concepts was measured by the survey. It was expected that secure programming self-efficacy would be strongly related to students' knowledge of secure programming. That knowledge was measured using the series of forty-five multiple choice secure programming questions as described above. Students generally scored poorly on the secure programming conceptual questions (M = 46%). This could be explained by the fact that for many students this was their first course in secure programming. However, both programming self-efficacy $r(65) = 0.34$, $p < 0.01$ and secure programming self-efficacy $r(65) = 0.35$, $p < 0.01$ were strongly correlated to performance (Table 2). Students who expressed high programming efficacy scored highly on the secure programming test.

Table 2. Correlation between secure programming knowledge, programming self-efficacy, and secure programming self-efficacy

	Knowledge	**Programming Efficacy**	**Secure Programming Efficacy**
Knowledge	1	0.349**	0.351**
Programming Efficacy		1	0.858**
Secure Programming Efficacy			1

**Correlation is significant at the 0.01 level

These results are in agreement with the theory that suggests that as students increase in knowledge, their self-efficacy in secure programming will also increase. This could be due to increased knowledge, practice, and exposure resulting in increased confidence among students. Students score higher in programming self-efficacy because they have more experience with programming than secure programming. However, programming self-efficacy is strongly related to secure programming self-efficacy because the latter is predicated on the former – i.e. students cannot develop knowledge in secure programming without prior knowledge of programming.

Students were also asked to report their level of expertise as regards programming. They were asked to rank their expertise as beginner, intermediate, competent, and expert. 31% of students reported intermediate expertise, 67% reported competent expertise, and 2% reported being an expert level. None of the students reported being beginners. When secure programming self-efficacy was broken down by reported expertise, we found that self-efficacy increased with expertise though their secure programming efficacy started lower and increased more significantly (Fig. 2). The pattern observed between performance and expertise is repeated between efficacy and expertise. This would suggest that self-efficacy may in fact be a mediating variable between expertise and performance.

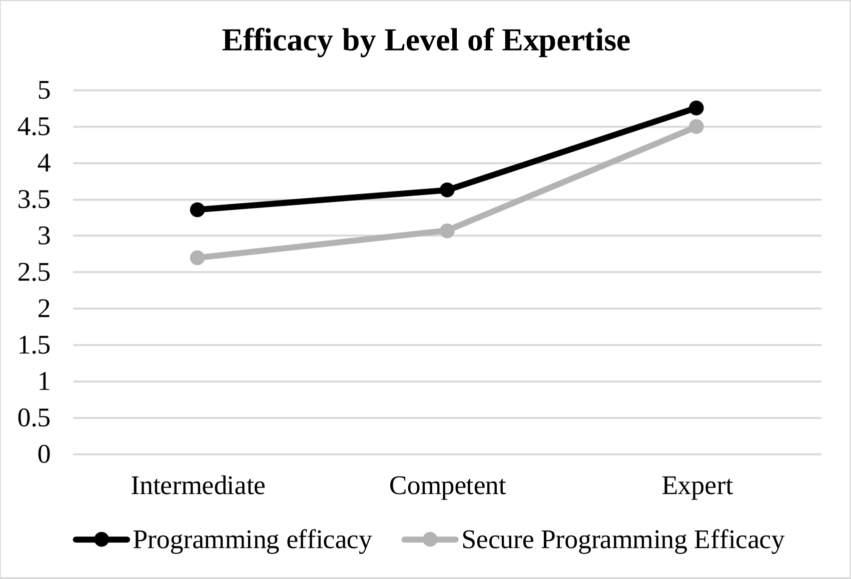

Fig. 2. Variation in self-efficacy by reported level of expertise

Table 3. Correlation among extracurricular activities, programming self-efficacy, secure programming self-efficacy, and secure programming knowledge

	Programming Self Efficacy	**Secure Programming Efficacy**	**Knowledge (%)**
Coding competitions	3.61*	3.02*	30.00
Programming message boards	3.55	3.07*	32.14
Programming internship	3.64*	3.03*	34.03*
Online gaming	3.5	2.92	33.63*

* Correlation is significant at the 0.01 level

Students develop expertise in various ways bringing prior knowledge from non-classroom activities. The survey asked students to report their engagement with a selection of programming related extra-curricular pursuits. 75% of students reported engaging online gaming, 72% reported having had a computing related internship, 42% reported asking questions and interacting with others on programming message boards, and 28% reported engaging in hackathons. These extra-curricular activities were selected because they were reported anecdotally as being the most common computing related activities among students. Coding competitions increased students' confidence in their programming ability but did not improve their performance on secure programming questions. Online gaming increased students' performance on secure programming questions but did not have an impact on self-efficacy. Participating in a programming internship had the greatest impact on students' expertise increasing their self-efficacy across the board as well as their performance on secure programming questions (Table 3).

As seen in Table 3, participating in programming competitions increased the students' confidence in their programming abilities, both general and secure, but did not correlate with increased performance on the knowledge questions. In fact, they performed notably worse. Participating with other students and practitioners on programming message boards increased secure programming efficacy, although not general programming efficacy, but did not have a noticeable effect on performance on secure programming questions. The most significant effect of prior experience was for those who had had programming internships. Unsurprisingly having had such an internship increased both programming and secure programming self-efficacy and performance on the secure programming questions. Perhaps the most surprising outcome was for those who participated in online games. The gamers reported lower self-efficacy levels for both secure programming and programming in general but performed notably better.

The results suggest that the SPC should not focus exclusively on secure programming skills. Secure programming is a subskill to programming. Interventions must therefore make sure to support students' programming skill and confidence as well. Various extra-curricular activities also support secure programming self-efficacy. This suggests that the SPC can use these avenues as a way to increase student skills. Incorporating elements like gamification and message boards into the clinic structure will provide alternate ways for students to improve.

5 Conclusion

This paper described the development of a secure programming self-efficacy scale with a programming self-efficacy subscale. The paper reported the validation process for the scale. The scale was found to have high internal consistency and load on two factors corresponding with the secure programming self-efficacy and the programming self-efficacy subscales. The paper also examined efforts to assure convergent and predictive validity of the secure programming self-efficacy scale by comparing the scale to measures. We found that secure programming self-efficacy is, as expected, positively correlated with programming self-efficacy, knowledge of secure programming, expressed expertise in programming, and experience with programming related extracurricular activities. The results show that self-efficacy results from practice and exposure to secure programming, which is consistent with the theory of the development of self-efficacy.

5.1 Limitations and Future Work

Ideally, the two studies should have followed the participants through their education at the institutions, but this was not possible. The Institutional Review Board approving the studies' protocols required that all participants in the study be anonymous. So each participating student was assigned a random number. In every assignment and interaction with the student, the student's name was replaced with the number. At the end of the term, the file containing the association of the student name with the random number was erased and deleted, so the identity of each participant could not be recovered.

The two studies reported in this paper collected data from computer science and computer engineering students at two public universities in the lower-Pacific in the United States. The educational systems in other states and countries differ, as does education at non-public institutions. Further, computer science students are generally male, reflecting the bias of the field of computer science [19,20]. Thus, this study should be rerun in other places with a more balanced population to better understand how differences in environment affect self-efficacy. The populations at these institutions were relatively small for a validation of this type. Women and other under-represented populations were also significantly underrepresented in our sample consistent with their representation in this population. Further studies to validate these instruments would test the scale against other proposed programming self-efficacy scales for stronger validation. These studies would also expand the population across the US and to other countries and over sample among women and under-represented populations.

6 References

1. CVE Database, https://cve.mitre.org. Last accessed 15 Apr 2021
2. Zetter, K.: Serious Error in Diebold Voting Software Caused Lost Ballots in California County—Update. Wired (8 Dec 2008)
3. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F. and Kohno, T.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In Proceedings of the 20th USENIX Security Symposium. USENIX Association, Berkeley, CA, USA (2011)
4. Weinberg, G.: The Psychology of Computer Programming, Van Nostrand Reinhold, New York, NY, USA (1971)
5. Bishop M., Orvis, B.: A Clinic to Teach Good Programming Practices. In Proceedings of the 10th Colloquium on Information Systems Security Education, pp. 168–174. (2006).
6. Bishop, M.: A Clinic for 'Secure' Programming. IEEE Security and Privacy **8**(2) pp. 54–56 (2010)
7. Dark, M., Stuart, L., Ngambeki, I., Bishop, M.: Effect of the Secure Programming Clinic on Learners' Secure Programming Practices. Journal for the Colloquium for Information Systems Security Education **4**(1) (2016)
8. Bishop, M., Dark, M., Futcher, L., van Niekerk, J., Ngambeki, I., Bose, S., Zhu, M.: Learning Principles and the Secure Programming Clinic. In Proceedings of the 12th IFIP WG 11.8 World Conference on Information Security Education (IFIP Advances in Information and Communications Technology 557) pp. 16–29 (2019)
9. Bandura, A.: Self-Efficacy: Toward a Unifying Theory of Behavioral Change. Psychological Review, 84(2) (1977)
10. LaMorte, W. The Social Cognitive Theory. (2019)
11. Bandura, A.: Self-Efficacy: The Exercise of Control. Worth Publishers, New York, NY (1997)

12. Ramalingam, V., Labelle, D., Wiedenbeck, S.: Self-Efficacy and Mental Models in Learning to Program. In ACM SIGCSE Bulletin **36**(3), 171–175. (2004)
13. Gurer, M., Cetin, I., Top, E.: Factors Affecting Students' Attitudes toward Computer Programming. *Informatics in Education* **18**(2), 281-296. (2019)
14. Ramalingam, V., Wiedenbeck, S.: Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. Journal of Educational Computing Research **19**(4), 367–381. (1998)
15. Sherer, M., Adams, C.: Construct Validation of the Self-Efficacy Scale. In Psychological Reports **53**(3), 899–902. (1983)
16. Chen, G., Gully, S., Eden, D.: Validation of a New General Self-Efficacy Scale. Organizational research methods **4**(1), 62-83. (2001).
17. Bishop M., Dai J., Dark M., Ngambeki I., Nico P., Zhu M.: Evaluating Secure Programming Knowledge. In: Bishop M., Futcher L., Miloslavskaya N., Theocharidou M. (eds) Information Security Education for a Global Digital Society. WISE 2017. IFIP Advances in Information and Communication Technology, vol 503. Springer, Cham.
18. Dark, M., Ngambeki, I., Bishop, M., Belcher, S.: Teach the Hands, Train the Mind . . . A Secure Programming Clinic! In: Proceedings of the 19th Colloquium for Information Systems Security Education. pp. 119–133. (2015).
19. Frieze, C., Quesenberry, J.: How Computer Science at CMU Is Attracting and Retaining Women. Communications of the ACM **62**(2), 23–26.
20. Ganley, C., George, C., Cimoian, J., Makowski, M.: Gender Equity in College Majors: Looking Beyond the STEM/Non-STEM Dichotomy for Answers Regarding Female Participation. American Educational Research Journal **15**(3), 453–487.