Learning Principles and the Secure Programming Clinic

Matt Bishop, Melissa Dark, Lynn Futcher, Johan Van Niekerk, Ida Ngambeki, Somdutta Bose, Minghua Zhu

Learning Principles and the Secure Programming Clinic

Matt Bishop[1][0000−0002−7301−7060], Melissa Dark[2], Lynn Futcher[3][0000−0003−0406−8718], Johan Van Niekerk[3,4][0000−0003−1739−4563], Ida Ngambeki[2], Somdutta Bose[1], and Minghua Zhu[1]

[1] University of California at Davis, USA {mabishop,sombose,mhzhu}@ucdavis.edu
[2] Purdue University, USA {dark,ingambek}@purdue.edu
[3] Center for Research in Information and Cyber Security, Nelson Mandela University, South Africa {lynn.futcher,johan.vanniekerk}@mandela.ac.za
[4] Noroff University College, Norway
johan.vanniekerk@noroff.no

Abstract. Several academic institutions have run a clinic on robust and secure programming. Each time a clinic was run, it was associated with a specific class. Using pre- and post-class evaluation instruments, it is clear that the effect of the secure programming clinic on students' understanding of secure programming was generally positive. However, in some instances the clinic was underutilized, and in other cases it could not be run at other institutions. The goal of this paper is to examine the structure of the clinic in light of five basic learning principles, and provide information about when a clinic will not improve students' understanding, and when it will. We validate this by examining an instance of the secure programming clinic, and show how the learning principles explain the improvement in student grades, or lack thereof. From this, we draw conclusions about ways to make the clinic more effective, and when it will not be effective.

Keywords: Secure programming clinic · learning principles · robust programming.

1 Introduction

The problem of nonsecure code is widely recognized as a major source of security problems. Indeed, of the vulnerabilities in the U.S. National Vulnerability Database in the last 5 years, over 19,000 are identified as injection and buffer overflow vulnerabilities, exemplars of poor programming practices [14]. Some, such as Heartbleed, have impacts throughout the Internet [7]. Industries, government, and many other organizations want programmers who can write secure, robust code. The problem is how to teach this material.

Computer-related majors, and many other majors, teach programming. Introductory classes typically teach good programming style, error checking, and other aspects of programming summarized by the word "robust". Being introductory, their focus is not on security as such, but simply on producing good programmers who understand both the strengths and limitations of the language they learn. Future classes expect the students to apply what they have been taught, and rather than focusing on style and robustness, the emphasis is upon correctness. Hence style and robustness is a minimal part of the grade, if they are included at all, giving students the impression that security and quality of programming is less important than required functionality. Students have no reason to apply what they learned in introductory programming. Further, advanced security issues such as programming to avoid race conditions is not taught in introductory classes, so the students lack knowledge of all the aspects of secure coding. The atrophying of this practice, and the lack of knowledge, leads to programmers who do not write robust, secure code.

This problem is exacerbated by the richness of the current curricular guidelines for computer science programs. The ACM Computing Curricula [23] is laden with more material than can be taught in a typical major program, so faculty and program developers choose which parts to emphasize. Given this, creating a required class on secure programming would require something else to be dropped from the curriculum. Further, most faculty do not know all the aspects of secure coding, so many institutions would need to find the instructors for these courses. Thus, few schools have any courses on secure programming, let alone required ones.

An alternative to additional classes is the Secure Programming Clinic [2, 3]. This clinic works analogously to an English writing clinic, and is discussed in detail in Section 3, below. Like any other aspect of education, its success depends upon a host of factors. The contribution of this paper is to use results from running several instances of this clinic, and basic educational principles, to identify characteristics of an environment in which the Secure Programming Clinic will enable students and instructors to ensure that students program in a robust and secure style, and that instructors who are not experts in this can have the clinic provide feedback about submitted programs that the faculty can take into account when assigning grades. The paper is structured as a Case Study according to guidelines for such studies provided by [6]. The next section provides a brief overview of the case study's layout and how it reflects in the structure of the remainder of this paper.

2 Methodology

The following structure is often used for the presentation of Case Study research [6]:

- Entry vignette
- Introduction
- Description of the case and its context
- Development of issues
- Detail about the selected issues
- Lessons Learned and Assertions
- Closing vignette

In the context of this paper, the abstract and introduction to the paper respectively serves as the case study's entry vignette and introduction. Section 3 describes the Secure Coding Clinics and serves as a description of the case and its context. The context is further supported by Section 4, which provides a review of relevant literature relating to the teaching of secure programming. The issues of concern in this paper are the identified educational principles, and how these were used in past secure programming clinics. These principles are identified in Section 5 and the relevance/adherence of the three variations of past clinics to the learning principles are mapped in Section 6 which elaborates on the selected issues of concern for this case study. These issues will be used to inform the authors' design intention for future secure programming clinics, which will be presented in the form of lessons learned in Section 6.2.

The closing vignette will take the form of our concluding remarks

3 Description of the Secure Programming Clinics

The basic form of a secure programming clinic is a physical or logical space where clinicians are available to students. As students complete homework assignments and other programs, they bring them to the clinic. The clinicians review the programs, and point out examples of code that create security problems or, more generally, are non-robust. They do not identify all such problems, but instead simply point out examples. They do not examine the program for correctness or whether it meets the requirements of an assignment. The students, on their own, correct the problems. The clinicians do not check that the program meets the requirements of the assignment; indeed, the clinic is not associated with any class, and is available on a drop-in basis. In this way, it resembles a writing clinic. Students can bring papers to the clinic, where clinicians will review the grammar and structure of the paper and offer suggestions on how to improve both. They do not examine the content for accuracy or credibility; they simply look at the form.

Several variations of the clinic are possible.

The first major set of variations comes from associating the clinic with a particular class. Here are possible approaches, any combination of which may meet the goals:

1. The instructor may apportion some part of the program grade to robustness. The clinicians would then assist by grading that part of the homework, and providing an appropriate score. The class graders would then grade the program with respect to the assignment's requirements.
2. After #1, the instructor can have the students correct the robustness problems, and then regrade that portion, giving the student some percentage of the points they corrected.
3. The clinicians can act as assistant instructors, helping the students develop threat models for how an attacker might use their program to violate desired security properties. As "security" is defined in terms of requirements, the threat model is critical to knowing the types of security problems that might

arise. On the other hand, robustness issues are independent of threats, in the sense that they are common to all threats.

The functions of the clinic can be extended beyond simply reviewing programs. It can also provide information to help the students fix the problems. This typically requires collecting examples of poor programming and how to fix, or (better) avoid, them. It can also provide remote assistance, where the clinicians are not at the institutions. There is a salutary effect for this. If some of the clinicians are volunteers who work in the software industry or for government agencies, their presence and activities will convey the importance that future employers place in high-quality code. This provides incentives for students to learn the material.

The clinic can also be shared among universities. One implementation of the clinic provides a common shared appointment calendar, so students from any of the academic institutions could sign up for appointments even when the local clinicians were not available. The clinicians from the institutions coordinated their times so that one was always available during the day. Were this to be extended internationally, clinicians would probably be available for most of the evening and night (when many students of computer science and related disciplines develop their programs).

The above discussion provides insight into the specific secure programming clinic format of concern to this paper. However, for the sake of comprehensiveness, the next section will briefly highlight other such approaches, and challenges, relating to the teaching of secure programming.

4 Teaching Secure Programming

Secure programming is about writing secure code. The focus of many programming courses, however, is to write code that works with a lack of focus on writing code securely. A developers unintentional ignorance of known vulnerabilities and insecure coding practices can generate insecure software. Besides the potential financial loss, the successful exploitation of insecure software can impact the confidentiality, integrity and availability (CIA) of critical information. Undetected exploitation can also lead to the embedding of malicious software within an organization, giving the malicious attacker the ability and potential to attack any time [18]. Secure programming should therefore include the basic principles of robust coding to guard against unexpected inputs and events [15].

The challenges of teaching and integrating secure programming into computing curricula have been around for many years, and some of these challenges which are still evident today [13]. These include:

- Lack of faculty buy-in
- Competition with other topics for inclusion into the curriculum
- Computing curricula already full
- Failure of students to grasp other important programming concepts
- Lack of secure programming expertise of faculty members

Much research has been conducted to address some of these challenges. A recent study [21] investigates a Java proof-of-concept plug-in for Eclipse, ESIDE (Educational Security in the IDE), that provides vulnerability warnings and secure programming education in the IDE while students write code. It works by scanning a selected project for code patterns that match predefined heuristic rules of security vulnerabilities. In this way, secure programming knowledge can be introduced early and reinforced throughout a students education. Generally, ESIDE was found to increase students awareness and knowledge of secure programming. However, almost no students actually modified their code to mitigate the detected vulnerabilities as they were most concerned with completing functionality and did not want to impact that functionality with additional security-oriented code. In addition, carefully timing the introduction of concepts and skills as well as incentivising such learning is important [21].

ESIDE was compared to the Secure Programming Clinic by running each approach with two separate groups of students, one group assigned to ESIDE and the other to the clinic [21]. Each group of students were asked to report on how likely they would use the recommended changes in their code during the session. The likelihood results for the Secure Programming Clinic were significantly better than for ESIDE. However, the clearest difference between the clinic and ESIDE were the number of specific vulnerabilities covered. Where ESIDE marked on average 42 lines of code per participant, the technical assistants running the clinic pointed out approximately two specific lines of code per participant.

One response to the need to teach students to program more securely was to introduce a serious game for teaching secure coding practices and principles to novice programmers [1]. Initial findings showed the game to be usable and engaging, with the majority of students being able to make clear correlations between the game levels and corresponding security concepts. Similarly, constructing secure coding duels [24] in Code Hunt, a high-impact serious gaming platform released by Microsoft Research, was proposed to instill gaming aspects into the education and training of secure coding. Secure coding duels proposed in this work are coding duels that are carefully designed to train players secure coding skills, such as sufficient input validation and access control. Using serious games for teaching secure coding could alleviate some of the challenges faced by faculty members in this regard.

Furthermore, scorecards and checklists provide a consistent means of evaluation and assessment [22] . They describe the use of security checklists and scorecards which provide a quantifiable list of security criteria to aid in writing secure code and further reinforce security principles. Checklists distributed to students included:

- Sample code of errors to look for;
- Examples of correct ways of writing code; and
- Security mantras including a list of principles that form the basis for the checklist, for example: All Input is Evil!

Regardless of the approach used to teach secure programming, such approaches should take into account recognized learning principles, as discussed in Section 5, to ensure that learning takes place.

5 Learning Principles

Systematic studies of human behaviour, including studies of how people learn, is a relatively new field of scientific enquiry [17]. However, despite the youth of this field, many studies have already been dedicated to investigating how learning takes place. During such studies, researchers strive to identify recurring patterns in the data and to make generalizations based on these patterns. Such generalizations lead to the formulation of *learning principles* and *learning theories*. Learning principles identify the factors that influences learning. For example, the *principle* that a behaviour which is rewarded in some way is more likely to re-occur in future than one which is not followed by a reward. A learning *theory* on the other hand aims to provide an explanation of the underlying mechanisms that are involved in learning. Thus, whilst a learning principle presents *what* factors are important, a learning theory would explain *why* those factors are important [17].

Learning *principles* do not change much over time, however, learning *theories* have continually changed as understanding of human behaviour evolved [17]. Due to the fact that learning principles are less changeable, and thus more 'future proof' than learning theories, this research will seek to identify learning *principles* that could be useful in the secure coding clinics, but will avoid subscribing to any specific learning *theory*.

Educational literature provides many such learning principles. These principles have been identified, and their impact verified, in a variety of ways. One such approach is the field of brain compatible education. This educational approach stems from a combination of neuroscience and educational psychology and was first made possible by advances in brain imaging during the 1990s [12].

Brain-compatible, or brain-based, learning is not a formalised education approach or 'recipe for teachers', instead it provides a "set of principles and a base of knowledge and skills upon which we can make better decisions about the learning process" [9, p xiii]. Brain research has shown that humans literally grow new dendrites and neural connections every time they learn something. Knowing which educational activities are the most effective in stimulating such growth allows educational practitioners to create material that leverages the way the brain naturally learns [10]. For the purpose of this research, it is not necessary to understand how these natural learning processes work. One only needs to understand that these principles were verified as being effective in promoting real learning.

No single complete list of such principles exists. However, many principles are presented and discussed in the literature [4,5,8,9,11,16,19,20]. The list presented in Table 1 contains a subset of principles from those used in literature. The principles included in Table 1 were restricted to those the authors specifically deemed most relevant to the context of the Secure Programming Clinic. Relevant principles from literature were reworded and consolidated in cases where there was significant overlap in meaning between those used in the literature and the context for use in this study. Table 1 thus presents the authors' adaption of these principles. The following discussion briefly elaborates on each of the listed principles:

– LP1 - According to [8,9,11] there is no long term retention without rehearsal. The brain would prune new neural growth if it is not reinforced by being used. It is vital to repeat lessons taught more than once, otherwise students would be likely to forget these lessons. One should also allow enough time for students to assimilate any new concepts. Several studies [4,8,11] explains that the brain will reconsolidate new neural growth for several weeks after learning using both conscious and unconscious (sleep) processes to decide how to incorporate knowledge into existing neural structures.
– LP2 - If the new knowledge is too advanced for the target audience, learning might be inhibited because the learners feel threatened instead of challenged by the content [4,5,8,9]. Furthermore, new knowledge can only be assimilated if it builds upon prior knowledge, since novel patterns can only form as extensions of existing patterns [5, 11, 19, 20].
– LP3 - The process of learning consists of the brain recognizing patterns [4, 5, 8, 9]. For these patterns to form the learners need to recognize and connect patterns by themselves [5, 9, 11, 19]. This process works best if the learners experience these patterns in contexts that are relevant to themselves [5, 9] and their real-life experiences [11].
– LP4 - Humans naturally learn in social settings and through interaction with others [4, 8]. Collaboration with others enhances learning [11].
– LP5 - Rehearsal will make learning permanent, however, this does not guarantee the rehearsed learning is in fact correct. Practice should be accompanied by *feedback* that is constant, consistent, and specific to ensure that practice that is permanent is also correct [8, 16]. The effect of feedback is also amplified if it is immediate [5, 9].

6 Mapping of Clinics to Selected Learning Principles

We begin by examining the instances of the secure programming clinic that have been run, and how they reflect the learning principles. We then discuss how the clinic might be improved by mapping the principles into various forms of the clinic.

6.1 Experimental Secure Programming Clinics

The University of California at Davis, the California Polytechnic State University at San Luis Obispo, the California State University at Sacramento, and Purdue

Table 1. Learning Principles

Principle#	Description
LP1	Lessons must be repeated at suitable intervals
LP2	Lessons must build upon the pre-existing knowledge of the target audience and must be of an appropriate level of difficulty
LP3	Learning happens through the recognition of patterns. To recognize new patterns, learning must be actively, personally, and specifically experienced in a context the learner can relate to
LP4	Learning is enhanced through collaboration and interaction with others
LP5	Immediate feedback amplifies learning

University Northwest have run the Secure Programming Clinic over the past 4 years. These instances of the clinic were tied to particular classes such as networking, operating systems, computer security, and introductory programming classes.The methodology was the same in all instances.

At the beginning of the term, students were asked to fill out an evaluation form that tested how much they knew about secure programming. They received class credit for beginning the questionnaire, and could indicate if they declined to proceed after giving their name and student ID (so they could get credit). At the end of the class, they filled out a similar questionnaire. The results of the two questionnaires were compared to see how their knowledge of, and ability to practice, secure programming changed. During the class, for each programming assignment, students could go to the clinic before submitting the assignment, and modify their programs based on the clinician's feedback. When assignments were submitted, they were given to the clinicians to check for the robustness and quality of the programming; graders assigned to the class graded the submitted programs for correctness. Then the instructor combined the results to give the program a grade. When the assignments were returned, students were told they could correct the robustness and security problems, and get back a large percentage of the points deducted for this (usually 75% or 80% of the points). They had a week to do this and resubmit the assignment. The clinicians would then review the changes, compare them with the original programs, and inform the instructor about the changes. The instructor would change the grade accordingly.

Institutional Review Boards (IRBs) at all institutions examined the experimental protocols to ensure they complied with federal and state law, and with the institutions' own rules about gathering and retaining student data. At the University of California at Davis, the principle investigator of the project was also the instructor, which the UC Davis IRB saw as a conflict of interest. To keep students anonymous to the instructor, student information (name and student identification number) was coded with a 4-digit number, and all work relating to that student's interaction with the clinic was recorded using that 4-digit number. During the term, the clinicians kept track of each student's 4-digit number, and any analyses had the student names and other identifying information redacted and replaced by the 4-digit number. When assignments were graded, the clinicians were given everyone's grade. This way, the instructor had no access to a particular student's pre- and post-questionnaires, nor to any information recorded about grade improvement.

Data on students' clinic usage and secure programming scores were collected in 2017 in one class (see Table 2). Of 42 students enrolled, 36 visited the clinic and 6 did not. Of the 36 students who visited the clinic, 14 students visited the clinic before the assignment was due, which were called "proactive" clinic users. Five visited the clinic before and after the assignment was due; they were called "consistent" users. Seventeen only visited the clinic after the initial secure programming assignment was submitted and graded; they were called "reactive" users.

The average score for 12 of these 14 students on the secure programming assignment was 77% and the standard deviation was 4%. These students did not submit their assignment for a regrade. Two of the 12 proactive students (students who visited the clinic before the assignment was due) submitted their original assignment, received a grade, and then made changes to their assignment and submitted their assignments for regrade. These two students differ notably. The first student received an initial score of 9% and did not gain any additional points when he/she submitted the assignment for regrade. The second student scored 78% on the initial assignment and 91% on the assignment that was submitted for regrade.

Table 2. Statistics from the Secure Programming Clinic.

		Initial clinic	Grade		Clinic	Regrade	
	n	visit	mean	stdev	revisit	mean	stdev
Proactive; no regrade	12	yes	77%	22%	no	N/A	N/A
Proactive; regrade	2	yes	44%	28%	no	50%[5]	41%
Consistent; no regrade	2	yes	34%	22%	yes	N/A	N/A
Consistent; regrade	3	yes	23%	10%	yes	77%	19%
Reactive; no regrade	0	no	N/A	N/A	no	N/A	N/A
Reactive; regrade	17	no	21%	17%	yes	70%	10%
Never; no regrade	0	no	N/A	N/A	no	N/A	N/A
Never; regrade	6	no	23%	98%	no	48%	16%

When the learning results of the Secure Programming Clinic are interpreted in the context of the 5 learning principles, there are several interesting findings. In the context of individual student learning, the principles are intersectional. For example, while lessons must be repeated at suitable intervals, what makes an interval "suitable" is partly contingent upon the pre-existing knowledge of the target audience, which is comprised of every student. Thus, pre-existing

[5] These data are anomalous. There were only two students in the group. One student scored 91% on the regrade and the other student remained at a score of 9%.

knowledge levels and learning trajectories vary. Another intersection is between LP1 and LP5. Immediate feedback (LP5) signals to learners the need for additional practice (LP1). When the learning setting provides for additional practice (LP1), at an appropriate level of difficulty (LP2) and supported through interaction with others (LP4), learning is supported.

Interpreted in the context of the principles, the 12 proactive clinic users who did not submit their assignments for regrade, visited the clinic. They then used the specific and timely feedback they were given to practice more secure programming. They submitted the assignment for feedback, and received timely positive feedback via their scores. This signals to the student that they are on track. In the case of the proactive student who received a high score and chose to make modifications to his/her assignment and submit for regrade, the clinic afforded the use of specific and timely feedback (LP5) in support of rehearsal and mastery (LP1 and LP2). In the case of the proactive student who received an initial score of 9% and did NOT improve, there are two considerations. If the student does not care about learning, then most instructional attempts at intervention will fail. However, if the student wants to learn, but is failing, then it would be important to incentivize the student to visit the clinic again. Provided there is desire to learn, the deficiency is likely due to prior knowledge (LP2) and/or lack of active, personally relevant experiences that afford pattern formation (LP3). In this case, the clinic should provide suitable diagnostics.

The 5 students who visited the clinic before the assignment was due and then after receiving an initial grade and before the reworked assignment was due are also interesting. Three of the five received a score of 23% on the robustness of their code for the assignment. It would seem as if the clinic was marginally useful for these three students before they did the assignment, especially if you compare them to the 12 proactive clinic users who scored 77% on the first assignment submission after visiting the clinic. Upon receiving their grade and then revisiting the clinic, these three students raised their grade to 77%. For these students, the value of the clinic demonstrates all of the learning principles. The clinic provided a venue for these students to repeat the exercises. The students seem to have acquired useful knowledge through this rehearsal, even though the new knowledge had to be built on wrong performance. Unlike their 12 peers in the proactive group who were able to attend the clinic, get useful information and submit the homework and earn a high grade, these students attended the clinic, got the proactive information, applied it incorrectly (LP2), received a poor grade (LP5), reattended the clinic (LP2, 3 and 4), learned about the same secure programming practices in the context of marginal performance (LP1, 2), and then succeeded. This is very encouraging.

The 17 students who were reactive users of the clinic are also interesting. Upon receiving an initial grade (mean score for this group was 21%, standard deviation 17%), learners had knowledge of what they did not know. The specific and timely feedback (LP5) acted as a spur to motivate students to use the clinic. The feedback was diagnostic in that it showed students exactly where their thinking was deficient, thereby allowing practice and rehearsal in correcting their insecure programming practices (LP1). In order for these students to continue expanding their secure programming practices, the clinic should support multiple classes, so that lessons are repeated (LP1), in a manner that subsequent knowledge builds on prior knowledge (LP2), culminating in the formation of patterns (LP3). When this is the case, learners will have robust knowledge of robust programming practices.

6.2 Discussion and Lessons Learned

The preceding section shows how important the principles are to the clinic's success. Here we discuss more general lessons learned.

Principle LP2 suggests that students build on pre-existing knowledge to learn, and LP3 says students learn by recognizing patterns. In the instance of the secure programming clinic for an introductory programming class in C, few students availed themselves of the clinic even though they were offered the chance to resubmit work, as described above. The problem was that the students were struggling to learn the language and techniques. Although all were supposed to have programmed before, none had used C, C++, or Java, and so concepts like pointers were new to them. They also had not mastered recursion and other basics. Hence they had no pre-existing knowledge of many features of C, and so they were unable to build on that pre-existing knowledge. Further, concepts of security were too difficult, so the clinic focused on robustness (avoiding buffer overflows, checking input, and so forth). Even these basics required the students to recognize patterns in programming, and they simply did not have the background for this.

The clinics discussed above were tied to specific classes. The students who went consistently (that is, before the assignment was due and again after the grading) showed the greatest improvement. This accords with learning principle LP1, that learning must be repeated at suitable intervals. Were the clinic not tied to specific classes, and were faculty to have their students use the clinic, this finding suggests that students would improve their robust and secure programming skills.

The structure of the clinic had the clinicians interacting directly with the students, sometimes in person privately, sometimes in a group, and sometimes using remote technology such as Skype. They gave immediate feedback by identifying examples of robustness and security problems in the students' programs, and discussing them, and approaches to possible solutions, with the students. The students then had not only to figure out the best solution and implement it, but they also had to analyze their program looking for other instances of that type of problem, because the clinicians did not point out all the problems. The goal was to give the students the skills to find the problems themselves, and learn how to avoid programming them in the future. This is an application of LP5, immediate feedback amplifying learning. Its success is shown by the marked increase in scores for the consistent and reactive students' programs.

The clinic provided students interaction with others (the clinicians). Although no formal statistics were gathered, the clinicians thought the group sessions were better because the students would explain problems to one another and therefore learn from their peers as well as from the clinician. This is an application of LP4, that collaboration and interaction with others enhances learning.

Several lessons from this mapping are apparent:

- The secure programming clinic will not work well with beginners or those who have little programming experience.
- The secure programming clinic will benefit students the most if it is available throughout the students' educational career, and they are required to use it in all classes that require programming (except introductory programming classes).
- The secure programming clinic must be tuned to the experience and background of the students, individually when possible, or taking into account the educational environment of the students when not.
- The secure programming clinic should hold group meetings as well as individual consulting sessions.

7 Conclusion

The secure programming clinic was developed to assist students in learning to write robust, secure programs, and to reinforce this throughout their schooling. A key question is to determine the conditions under which it will work well, and under which it will not work well. The research presented above, and its interpretation in light of the educational principles, provide general answers.

An interesting question would be to determine how to tailor the secure programming clinic to take advantage of specific educational environments. As an example, the clinic would probably be instantiated differently in South Africa than in the U.S. due to the differences in the structure of the academic programs. How to take those differences into account, to make the clinic maximally effective, is ripe for study. Similarly, could the clinic be adapted to a short, week-long course designed to teach programming intensively, and if so, how would the clinic operate to provide support after the course? Any analysis or development of these clinics must examine how to apply the principles in light of the environmental constraints.

Acknowledgements. This material is based upon work supported by the U.S. National Science Foundation under Grant Number DGE-1303211 to the University of California at Davis and Grant Number DGE-1303048 to Purdue University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. National Science Foundation, the University of California at Davis, and Purdue University.

References

1. Adamo-Villaani, N., Oania, M., Cooper, S.: Using a serious game approach to teach secure coding in introductory programming: Development and initial findings. Journal of Educational Technology Systems **41**(2), 107–131 (Dec 2012). https://doi.org/10.2190/ET.41.2.b
2. Bishop, M.: A clinic for 'secure' programming. IEEE Security and Privacy **8**(2), 54–56 (Mar 2010). https://doi.org/10.1109/MSP.2010.62
3. Bishop, M., Orvis, B.J.: A clinic to teach good programming practices. In: Proceedings of the 10th Colloquium on Information Systems Security Education. pp. 168–174 (Jun 2006), https://www.cisse.info/resources/archives/file/68-s05p05-2006?tmpl=component
4. Caine, R.N., Caine, G.: Making Connections: Teaching and the Human Brain. Association for Supervision and Curriculum Development, Alexandria, VA, USA (1991)
5. Craig, D.I.: Brain-compatible learning: Principles and applications in athletic training. Journal of Athletic Training **38**(4), 342–349 (Oct 2003), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC314395/
6. Creswell, J.W.: Qualitative Inquiry and Research Design: Choosing Among Five Approaches. SAGE Publications, Thousand Oaks, CA, USA, third edn. (2012)
7. Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., Halderman, J.A.: The matter of heartbleed. In: Proceedings of the 2014 Conference on Internet Measurement Conference. pp. 475–488. IMC '14 (Nov 2014). https://doi.org/10.1145/2663716.2663755
8. Fogarty, R.: Brain-Compatible Classrooms. Corwin, Thousand Oaks, CA, USA, third edn. (2009)
9. Jensen, E.: Brain-Based Learning: The New Paradigm of Teaching. Corwin, Thousand Oaks, CA, USA, second edn. (2008)
10. Lombardi, J.: Beyond learning styles: Brain-based research and english language learners. The Clearing House: A Journal of Educational Strategies, Issues and Ideas **81**(5), 219–222 (jun 2008). https://doi.org/10.3200/TCHS.81.5.219-222
11. Materna, L.E.: Jump-Start the Adult Learner: How to Engage and Motivate Adults Using Brain-Compatible Strategies. Corwin, Thousand Oaks, CA, USA (20087)
12. McGeehan, J.: Brian-compatible learning. Green Teacher **64**, 7–13 (2001), http://www.bbbforlearning.com/uploads/1/0/4/4/10446722/brain_-compatable_learning.pdf
13. Nance, K., Hay, B., Bishop, M.: Secure coding education: Are we making progress? In: Proceedings of the 16th Colloquium for Information Systems Security Education. pp. 83–88 (2012), https://www.cisse.info/resources/archives/file/299-p13-2012?tmpl=component
14. National Institute of Standards and Technology: National vulnerability database, https://nvd.nist.gov
15. Ngambeki, I., Dark, M., Bishop, M., Belcher, S.: Teach the hands, train the mind ... a secure programming clinic. In: Proceedings of the 19th Colloquium for Information System Security Education. pp. 1–15 (Jun 2015), https://www.cisse.info/resources/archives/file/359-p10?tmpl=component
16. van Niekerk, J., Webb, P.: The effectiveness of brain-compatibkew blended learning material in the teaching of programming logic. Computers & Education **103**, 16–27 (Dec 2016). https://doi.org/10.1016/j.compedu.2016.09.008, https://www.sciencedirect.com/science/article/pii/S036013151630166X

17. Ormrod, J.E.: Human Learning. Pearson Education, Boston, MA, USA, sixth edn. (2011)
18. Paul, M.: The need for secure software. Harvard Business Review (2012)
19. Smilkstein, R.: We're Born to Learn: Using the Brain's Natural Learning Process to Create Today's Curriculum. Corwin, Thousand Oaks, CA, USA, second edn. (2011)
20. Sousa, D.A.: How the Brain Learns. Corwin, Thousand Oaks, CA, USA, fifth edn. (2016)
21. Tabassum, M., Watson, S., Richter, L.H.: Comparing educational approaches to secure programming: Tool vs. ta. In: Proceedngs of the 13th Symposium on Usable Privacy and Security. SOUPS 2017, USENIX Association, Berkeley, CA, USA (2017), https://www.usenix.org/conference/soups2017/workshop-program/wsiw2017/tabassum
22. Taylor, B., Azadegan, S.: Using security checklists and scorecards in cs curriculum. In: Proceedings of the 11th Colloquium for Information Systems Security Education. pp. 82–87 (Jun 2007), https://www.cisse.info/resources/archives/file/85-s05p01-2007?tmpl=component
23. The Joint Task Force on Computing Curricula: Computing curricula 2001 computer science. Journal on Educational Resources in Computing **1**(3es) (Fall 2001). https://doi.org/10.1145/384274.384275
24. Xie, T., Bishop, J., Tillmann, N., de Halleux, J.: Gamifying software security education and training via secure coding duels in code hunt. In: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. pp. 26:1–26:2. HotSoS '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2746194.2746220, http://doi.acm.org/10.1145/2746194.2746220