Teach the Hands, Train the Mind … A Secure Programming Clinic!

Melissa Dark, College of Technology, Purdue University
Ida Ngambeki, College of Technology, Purdue University
Matt Bishop, Dept. of Computer Science, University of California at Davis
Steven Belcher, National Security Education

Introduction

One of the major weaknesses in software today is the failure to practice defensive or secure programming. Most training programs include only a shallow introduction to secure programming, and fail to integrate and emphasize its importance throughout the curriculum. Yet the community advocates for the inclusion of good coding practices into the teaching and practice of programming in learning institutions. This begs the question; Shouldn't we teach those who program to use robust coding practices from the beginning of writing programs, rather than the failed strategy of making programs robust after they are written?

Considerable pressure has been building to do this; perhaps most telling are the two most recent Cybersecurity Acts proposed in Congress. The Cybersecurity Act of 2010 (S. 773, Titles I, 11(a)1(c) and II, §302(c)) and the Cybersecurity Act of 2012 (S. 2105, Title V, §501(d)) contain substantially similar language requiring that Congress receive reports assessing "secure coding education in colleges and universities". The National Cybersecurity Workforce Framework – a report that aims to improve the ability of academia and public and private employers to prepare, educate, recruit, train, develop, and retain a highly-qualified cybersecurity workforce. The Framework calls for improved software assurance and security engineering and further specifies that graduates need to develop "new (or modify[ing] existing) computer applications, software, or specialized utility programs following software assurance best practices" [NICE12, p. 13]. With the recent calls for improved practices in robust programming and for improvements in software assurance education so clear, the timely and relevant question is, how?

Academic institutions teach some secure programming in introductory classes, but often by the time students enter advanced courses, the teachers have only enough resources to focus on the correctness of code. Ancillary properties, such as robustness and security, are overlooked by necessity. Three basic issues underlie the problem of teaching students how to write secure code: the focus of introductory programming courses, the assumption that students will apply learned techniques of good programming in future work, and the lack of room in the computer science curriculum to add more material.

First, beginning programming classes typically focus on algorithmic and language issues rather than environmental issues. These classes teach some elements of secure programming, such as good program structure, basic input validation, checking bounds for array references and checking that pointers are non-null. They do not teach more advanced elements, such as avoiding race conditions and authentication over a network, because those elements involve knowledge that a beginning programming student is not

expected to have. These classes, if well taught, lay a foundation for secure programming techniques.

Second, classes after the introductory programming class assume that students know, and will apply, principles of good programming. In practice, this is not true. Students tend to focus on what is being taught, and regard the programs they write as instruments to exercise that knowledge. This is appropriate, but—like an English essay comparing Orwell's *1984* to Huxley's *Brave New World*—the expression of the content is as important as the content. In other words, if the program is poorly crafted, the student may convey that he or she understands the material, but the program may interfere with that demonstration. Unfortunately, graders (and many teachers) ignore the issue of well-crafted, robust code when they grade, and simply check that the program works. This gives little to no reinforcement of the importance of the techniques of robust programming, and few, if any, rewards for avoiding poor programming, in these classes.

Third is the ongoing debate of where to teach elements of secure programming. Should a separate class cover the material, or should it be integrated with existing classes? These approaches have advantages and disadvantages. A separate class allows the student and instructor to focus on *why* these techniques are important, what happens when they are not applied, and to explore the issues in more depth. But students have to take the class to benefit, which often is not the case since many are elective. Integrating the material into existing classes ameliorates this problem, but it also adds a burden to those classes. They must now cover more material, and instructors must write the material and integrate it into what they teach. Both these methods encounter the same problem. A review of the ACM Computing Curricula [ACM01] shows how much material must be compressed into courses for computer science majors. The focus of courses, naturally enough, is on the material intrinsic to the course and not to ancillary issues. Students also reflect this belief. Most teachers who deduct points for non-robustness or poor programming have heard the protest, "But it works!"

In 2011, the National Science Foundation sponsored a meeting, the Summit on Education in Secure Software. The Final Report [BuBi11] examined a number of ways to deal with this situation; one in particular, a "secure programming clinic" approach, does not require adding new courses, and can in fact be integrated into existing curricula. A "secure programming clinic," analogous to a writing clinic in law schools or English departments, provides continual reinforcement of the mechanisms, methods, technologies, and need for programming with security and robustness considerations throughout a student's undergraduate coursework. The clinic augments courses, not replaces them or their content.

The concept of a secure programming clinic is grounded in providing practical educational training for students that extends and reinforces the theory they learn in classes. Practical training can be instantiated through various means including field experience, internships, clerkships, clinical experiences, and the like. The general purposes for practical experience in the curriculum are to: 1) link theory to practice by providing regular and structured opportunities for students to apply and test knowledge

and skills; 2) raise problems and issues which are used to trigger the investigation of related theory and knowledge; and 3) turn learning into experience and experience into learning, thereby enabling learners to gain mastery of content.

Toward this end, this project is designing, developing, implementing and testing a Secure Programming Clinic (SPC). The SPC will use the principles of clinical education to provide students with a context-based experience. We believe that it might be a very effective approach for 1) inculcating secure programming into a 124+ credit hour degree program (typical BSCS) that aims to graduate students in a reasonable amount of time (i.e., 4-5 years) and 2) promoting expertise in secure programming within learners.

Levels of Knowing: Novice to Expert

A. Novice-Expert

Early work [DrDr80] on the novice-expert continuum identified fives levels of advancing competence: novice, advanced beginner, competent, proficient, and expert. Dreyfus and Dreyfus identified the following characteristics of novices: 1) adherence to rules, 2) attempting tasks with little to no strategy, 3) organizing a problem in vague or random ways, and 4) a lack of discretionary judgment. A more recent study [SpSt00] identified a four-stage developmental trajectory from novice to expert using two dimensions: competence and consciousness. The four stages are: 1) unconscious incompetence, 2) conscious incompetence, 3) conscious competence, and 4) unconscious competence. Novices posses unconscious incompetence meaning they both lack skill and awareness of what they need to learn. As they gain knowledge, they become more aware of what skill they lack and what they need to learn. As mastery develops, learners exhibit more competence in the domain, but skill must self-regulate their own learning. Finally, experts function in a manner where they exercise the skills in their domain proficiently and with a degree of instinctiveness and automaticity. The proficiency that experts exhibit is not heavy reliance on rules, but rather judgment based on deep, tacit understanding. Experts are able to relate domain specific objects and recognize complex patterns [RCSS01]. Experts are able to recognize different elements of a problem, integrate them and map them more accurately to the relevant knowledge systems [BaDa92] [KaGe02]. With expertise comes the confidence to organize knowledge in uncommon ways leading to a wider range of potentially novel solutions. Due to their well-organized knowledge networks or mental models, experts are also able to solve problems more quickly, and transfer what they have learned from one situation to the next.

B. Developing "Expertise" and the Secure Programming Clinic (SPC)

Naturally, One thing educators are interested in is the type of learning activities and environments that facilitate the progression from novice to expert. Here we elaborate on the nature of expertise and tie that to five foundational design implications for the SPC.

The educational research literature on developing "expertise" does not naively assume that educational programs can develop "experts" rapidly, or through a unidirectional

transfer of information. The development of expertise requires time, repetition, and the accumulation of a large store of knowledge and patterns and the ability to retrieve and recombine these and apply them in new situations [ChSi73] [KBNL11]. The development of expertise is an interactive process that requires learners to be active participants in a community of practice [BBC00].

Instructional Design Implication #1: Community of Practice - the SPC will provide this community of practice by connecting students to each other and to a range of subject matter experts (SMEs) in the field of secure programming who will serve as mentors. Through SME participation in this community, novices are exposed not only to the knowledge and language of a domain, they also learn the framework and history of that knowledge allowing them to properly contextualize it and form more organized and comprehensive mental models.

While it is important for those with less experience to have access to a range of SMEs when learning, it is important that the expertise be used purposefully. The very nature of expertise can make experts inadvertently be an obstacle to effective teaching and learning. Because experts are able to facilely integrate material and instinctively process information, they might take leaps that fail to explicate content to novices. Novices often need focused practice and feedback on component knowledge and skills, as well as practice integrating knowledge into the larger whole.

Instructional Design Implication #2: Scaffolded Learning – the SPC will scaffold learning by breaking down complex material into component parts, and encouraging targeted practice where appropriate. The SPC will also use whole-part strategies to help learners practice and master fluent integration of more complex knowledge and skills.

One of the primary distinctions between experts and novices is in the organization of their mental models. Experts have well-organized knowledge networks [GoSi98]. Novices on the other hand, generally display poorly organized knowledge networks characterized by missing or dislocated information. Novices often fail to make critical connections, or conversely, connect concepts that are not related, usually because of misconceptions about how new knowledge relates to prior knowledge. It is here that concept mapping can play an important role. Use of a concept map can help novices to understand the relationships amongst various concepts and therefore help them build more robust mental models.

Instructional Design Implication #3: Mental Models - the SPC will use concept maps both to help students understand the connections amongst concepts, and to measure the development of their mental models.

Experts accumulate knowledge over numerous learning experiences, many of which are not traditional classroom experiences. Several studies of computer programmers found that expert programmers were more able to recall large sections of code because they understood the function of the code and the principles that governed the relationships amongst functions [Bar86] [GuMa90]. These experts reported developing this expertise largely from writing and reviewing hundreds of pages of code rather than from narrow

classroom experiences. Classroom experiences, however, can be very important in laying the foundations of knowledge networks and addressing misconceptions.

Instructional Design Implication #4: Numerous Learning Experiences - the SPC will give the students access to a large sampling of code provided by the expert mentors, partner organizations, and existing repositories, and opportunity to build, check, and refine.

Another key distinction between novices and experts is in the level of abstraction of their knowledge. Experts display knowledge networks ranging in levels of abstraction from specific knowledge, which only applies under specific conditions, to abstract knowledge, which can be applied to general situations [AnFe08]. This abstracted knowledge is based on principles and is usually derived from repeated learning at the contextual level where the need for abstraction is designed into the problem, thus creating the potential for transfer. For novices, on the other hand, knowledge is closely connected to the conditions in which it was learned. Novices tie principles and concepts that they know to the surface features of how they were taught the principle or concept; when the context changes, they often fail to transfer what they have learned to make it applicable in the new context. It is therefore extremely important to provide students with learning experiences that allow them to solve problems in different contexts. This will allow them to learn to differentiate between context dependent information and "principles", which can be transferred across different contexts [BBC00].

Instructional Design Implication #5: Levels of Abstraction - the learning experiences provided by the Secure Programming Clinic will not only be numerous, but will provide the necessary diversity of experiences and contexts to help the learners abstract their developing knowledge across contexts.

Applying the Instructional Design Implications - Proposed Clinic Structure

The SPC will therefore be designed to provide a community of practice within which students will have numerous learning experiences both, scaffolding component knowledge and skills, and formulating knowledge at different levels of abstraction, to help students develop expertise through the building of correct and robust mental models.

The SPC will be built and expanded gradually over the next four years. Initially it will be staffed by two graduate students with extensive experience in secure programming, each working twenty hours a week. In the first few iterations it will be directly related to specific courses. Students taking these courses will be required as part of 2-5 assignments to submit their programs to the SPC for review and feedback. Students will email their programs to the clinician and schedule a consultation. During this consultation, the clinician will discuss the robustness of the program and make suggestions for improvement. Clinicians will then assess the improvements made to the program and provide feedback to the instructors who will include this feedback as part of the grade for the assignment. These assignments and interactions between students and clinicians will provide students with additional experiences that emphasize the importance of secure

programming. These interactions will also serve as the beginning of a community of practice.

As the SPC matures the clinical education model will be utilized to a greater extent. The clinic will therefore be structured like a residency program; students will learn theory, practice under supervision, then have some autonomy to train others. Therefore, the SPC will have expertise at three levels: expert clinicians, who will be volunteers, recruited from academia and industry; proficient clinicians who will be senior undergraduate and graduate students who have demonstrated skill with secure programming; and students who are the primary audience for the clinic. The expert clinicians will serve as mentors and will be available electronically at various dedicated times to interact with students answering questions and giving mini workshops on secure programming. These expert clinicians will also provide samples for the students to work on so they can see how secure programming could apply in different contexts. The proficient clinicians will be graduate and senior undergraduate students who will be "staffing" the clinic either as volunteers, as paid student workers, or in return for course credit. They will be available electronically or in person at regular times to discuss programs from the clinic bank, to provide feedback to students on specific programs, and to grade students' work for particular courses. The students will learn from both the proficient and expert clinicians and will have the opportunity to serve as clinicians themselves either by helping their peers by providing feedback on programs or by eventually joining the clinic as proficient clinicians. Students will also have the opportunity to contribute to the SPC by submitting copies of their own programs to the website or creating programs with intentional errors that others could use as learning tools. The SPC will also have a steering committee consisting of the primary researchers on the project and the project advisory board. This steering committee will design the SPC, plan clinic activities and evaluate both the performance of the clinic and student learning as a result of involvement with the clinic.

The SPC will be both a physical and virtual space so students will have access to the clinic through appointments and drop-ins and will also be able to interact with their peers and access the website. The web site will provide examples of non-secure and non-robust programming, as well as an explanation of the problem and the way to write the code robustly and securely. The target audience of the examples will vary. Basic robust issues, such as checking the length of data entered into a buffer, basic input validation, and other issues normally presented in an introductory programming class, will be aimed at beginning programmers. More advanced issues, such as race conditions and input validation on the web (to prevent cross-site scripting and SQL injection), will be aimed at students with a background sufficient to know the basic concepts of parallel processing (race conditions) and networking (validation on the web). Note this differs from existing “secure programming” web sites that are written for advanced programmers, or that provide exercises in writing such code [TK11]. The sources of such examples will be either real programs or code snippets from the Juliet suite available at the Samate area of the NIST web site. As these pages will be available to anyone, students can study them while writing programs, or the clinicians can use them to supplement their interactions with the students. It also enables clinics separated geographically to pool resources to aid students.

This proposed SPC structure fulfills all five design principles. The interactions amongst students and clinicians creates a community of practice that will help students to learn secure programming language, concepts, and skills within the context of the history and current framework of the field. These interactions, coupled with access to the website provide both numerous learning experiences and differing levels of abstraction to help students move from simply memorizing the principles to a deep understanding of the concepts and the ability to evaluate programs and create appropriate robust programs in different contexts. Finally, the multiple levels of feedback, coupled with the opportunity to view multiple examples, learn from experts and practice consistently will help students build well organized knowledge networks.

Table 1: Description of SPC structure

Title	Role	Source	Remuneration	Responsibilities
Expert clinician	Primarily teaching	Experts from academia, government, and industry	None - volunteer position	Mentoring, answer programming and design questions, teach mini-workshops, discuss career opportunities, dissemination of clinic.
Proficient clinician	Teaching and learning	Senior undergraduate students, graduate students	Volunteers, credit, paid	Mentoring, answer programming and design questions, teach mini-workshops, grading.
Student	Primarily learning	Students at the university	None	Submit programs for review and critique.
Instructor	Evaluation	Instructors at the university	None	Assign grades to student assignments, possibly based on evaluations supplied by proficient clinicians.
SPC steering committee	Management and Evaluation	Experts from industry, academia	None, consulting fee	Planning and management of clinic activities, recruit clinicians, evaluation of clinic

				performance, dissemination

Evaluating the Clinic

Assessment will be conducted in order to 1) assist learning, 2) measure individual achievement, and 3) evaluate the SPC. Assessment data will be used to make improvements to student learning and to the SPC, as well as to report outcomes in achievement as well as overall effectiveness of the SPC. The evaluation plan for the SPC consists of two primary activities 1) Assessing the knowledge gains and 2) Assessing indication of evolution from novice to advanced beginner, from advanced beginner to competent, from competent to proficient, and from proficient to expert. These evaluation purposes and activities lead to the following evaluation design implications.

Evaluation Design Implication #1: assessment will provide informative and timely feedback to learners because practice and feedback are critical to the development of knowledge and skills building into knowledge networks.

Evaluation Design Implication #2: assessment will need to include methods to evaluate component skills and discrete bits of knowledge, as well as abilities to integrate knowledge and skills into more complex models. This evaluation should include how learners 1) organize acquired information, 2) recognize patterns, 3) retrieve information, and 4) apply knowledge.

Evaluation Design Implication #3: assessment needs to examine how well students engage in practices appropriate to the secure programming domain, what they understand about those practices, and how well they use the tools and knowledge appropriately within the domain.

Evaluation Design Implication #4: assessment needs to carefully consider learners' ability to undertake near and far transfer.

Evaluation Design Implication #5: assessment should evaluate what schemas students are developing and under what circumstances. This includes both static, one-time depictions of mental models and time 1, time 2, and time n, as well as dynamic depictions showing structural changes that help us understand developmental pathways from novice to expert.

Evaluation Design Implication #6: assessment needs to provide explanatory power that links students' knowledge and evolution from novice to expert in the context of the design and implementation of the SPC. This evaluation information will help inform modifications to improve the clinic, as well as dissemination of the effectiveness of the SPC model to other educators.

Challenges to Implementing the Clinic

There are several challenges that still need to be addressed in the design of the SPC.

Quality control - It will be necessary to continuously evaluate the performance of the clinicians to ensure that students are receiving appropriate and helpful feedback. However, given the structure of the clinic where clinicians have a great deal of autonomy, exist in a peer network rather than a hierarchical network, and most of the interaction is virtual and anonymous, it will be impossible to evaluate all clinician-student interactions to ensure that feedback is continuously high quality.

Plagiarism - Students will be able to use the SPC to see program samples from the website, get feedback and input from clinicians and their peers, and see other people's programs. They might be tempted to have others complete their work or appropriate solutions from their peers.

Intellectual property - The SPC will encourage students to submit samples of their own work to the website so others can learn from their mistakes and their successes. However, all submissions will be anonymous and accessible to all users of the SPC making it difficult to retain ownership of intellectual property.

Propagating the clinic - One of the major goals of this program is to expand the use of secure programming clinics at other institutions. Scaling the clinic to cover multiple universities can be done in two ways. First, each school can have its own set of clinicians. These will be drawn from students and external volunteers. Second, several schools can pool volunteers and others to provide a central clinic that will advise students remotely. In either case, all clinics will be asked to develop pages for their own web sites, and the clinics will share the pages. This way, each school can tailor their own web site to their specific needs when necessary, and simply use other, existing web pages when such tailoring is unnecessary.

Conclusion

Many computer science programs fail to provide students with a comprehensive education in secure programming because 1) advanced understanding of secure programming requires advanced programming knowledge so secure programming cannot be taught in one introductory course 2) the importance of secure programming is not emphasized beyond basic principles 3) overloaded curricula mean that secure programming courses are often elective, if offered at all. We propose the use of a Secure Programming Clinic (SPC) to help address this lack while integrating secure programming across the curriculum without adding to the course load. This paper describes how a Secure Programming Clinic could be structured and evaluated.

References

[ACM01] ACM Computing Curricula 2001: Computer Science, Association for Computing Machinery, New York, NY (Dec. 15, 2001).

[AnFe08] D. Andre, & G. Fernand, G. Sherlock Holmes—An Expert's view of Expertise. *British*

Journal of Psychology, 99, 109 –125. (2008).

[Bar86] W. Barfield, Expert-novice Differences for Software: Implications for Problem-solving and Knowledge Acquisition. *Behaviour & Information Technology*, *5*(1), 15-29. (1986).

[BaDa92] D. Batra, & J.G. Davis, J. G. Conceptual Data Modeling in Database Design: Similarities and Differences between Expert and Novice Designers. *International Journal of Man-Machine Studies*, *37*(1), 83-101. (1992).

[BBC00] J.D. Bransford, A.L. Brown, & R.R. Cocking, *How People Learn—Brain, Mind, Experience, and School*, National Academy Press, Washington DC. (2000).

[BuBi11] D. Burley, & M. Bishop, M. *Summit on Educational in Secure Software, Final Report,* June 30, 2011, Report GW-CSPRI-2011-7, Technical Report CSE-2011-15. (2011).

[ChSi73] W.G. Chase, & H. Simon, Perception in Chess. *Cognitive Psychology*, 4, 55– 81. (1973).

[DrDr80] S.E. Dreyfus, & H.L. Dreyfus, Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition. Washington, DC: Storming Media. (1980).

[GoSi98] F. Gobet, & H. Simon, Expert Chess Memory: Revisiting the Chunking Hypothesis. *Memory,* 6, 225–255. (1998).

[GuMa90] B. Guerin, & A. Matthews, The Effects of Semantic Complexity on Expert and Novice Computer Program Recall and Comprehension. *The Journal of General Psychology*, *117*(4), 379-389. (1990).

[KaGe02] M. Kavakli, & J.S. Gero, The Structure of Concurrent Cognitive Actions: A Case Study of Novice and Expert Designers. Design Studies, 23(1), 25-40. (2002).

[KBNL11] K. Kim, J. Bae, M. Nho, & C.H. Lee, How Do Experts and Novices Differ? Relation Versus Attribute and Thinking Versus Feeling in Language Use. *Psychology of Aesthetics, Creativity, and the Arts*, 5 (4), 379-388. (2011).

[MaWe86] R. Spencer Mason, & R.W. Weisberg, Context-dependent Efforts on Analogical Transfer. *Memory and Cognition,* 14 (5), 442-449. (1986).

[NICE12] The National Cybersecurity Workforce Framework, National Initiative for Cybersecurity Education (2012); available at http://csrc.nist.gov/nice/framework/

[RCSS01] E.M. Reingold, N. Charness, R.S. Schultetus, & D.M. Stampe, D. M. Perceptual Automaticity in Expert Chess Players: Parallel Encoding of Chess Relations. *Psychonomic Bulletin and Review*, 8, 504 –510. (2001).

[SpSt00] J. Sprague, & D. Stuart, The Speaker's Handbook Fort Worth, TX, Harcourt College Publishers. (2000).

[TK11] B. Taylor and S. Kaza, "Security Injections: Modules to Help Students Remember, Understand, and Apply Secure Coding Techniques," *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* pp. 3–7 (2011).