Provisioning Secure Coding Curricular Resources: Toward Robust Software

Barbara Endicott-Popovsky, *IEEE*, Sam Chung, *Center for Information Assurance & Cybersecurity, University of Washington*, Viatcheslav Popovsky, *University of Idaho*

Abstract –The same vulnerabilities continue to appear in code, over and over again, yet many educational institutions continue to teach programming as they always have. Some high-tech companies have found it necessary to establish ongoing security training for their developers to make up for the absence of college-level, secure coding curriculum. Recently, the thread model, which integrates security concepts into existing computing curricula, has been recognized as effective to transform education in secure software, while not impacting resource-limited institutions with a complete curriculum change. Using this thread approach, we developed curricula inserts that included a programming assignment using a threat modeling tool, a design assignment applying a secure software development life cycle, a study comparing non-secure with secure code, and a re-documentation technique that produces secure code from non-secure programs. We introduced these curricula resources during a secure coding workshop for instructors. Their responses to assessment surveys provide insight into how to provide faculty support for the inclusion of security content in existing courses through the proposed curricula resources.

Index terms – Secure Coding, Thread Teaching Model, Pedagogical Model, Cybersecurity Body of Knowledge, Instructional Media

I. INTRODUCTION

It is estimated that 90 percent of reported security incidents result from exploits against defects in the design or code of commonly used software [1]. According to Symantec's vulnerability trend analysis, the total number of vulnerabilities is on the rise, from 4,814 in 2009 to 6,253 in 2010—a 30% increase [2]. By improving the education of computer scientists to include secure coding practices, we could expect significant reduction in the number of software vulnerabilities produced in code.

There have been three well-documented approaches to teaching secure coding techniques [3, 4]: 1) the single-course approach, 2) the track approach, and 3) the thread approach. The single-course approach is as its name implies—the introduction of a single course on secure coding practices, generally at the end of an undergraduate program. The track approach is similar. Several additional courses, instead of just one, are added to existing curriculum, to create a concentration that provides a more in-depth understanding. The thread approach, in contrast, recommends integration of security concepts across existing Computer Science (CS) and Information Systems (IS) curriculum.

The thread approach has been recognized as pedagogically more effective, while at the same time not impacting resource-limited institutions unnecessarily with the overhead of making a complete curriculum change. Adopting a thread approach, institutions need only a small budget to upgrade curriculum to include secure coding concepts, and faculty members need only to spend a small amount of time to make needed changes [4]. There is no need to introduce completely new courses that require a lengthy internal curriculum review process that may slow implementation. Several successful attempts at the thread approach have been reported [4, 5].

In spite of reported success, many faculty members find it too time consuming to make the needed curricular improvements. Others are unsure about how to incorporate secure coding concepts into existing courses. Still others are simply unaware.

For this reason, we designed a two-day secure coding workshop for faculty. The content included: 1) a comprehensive pedagogical model that provides a tool for adjusting curriculum to student audience [1, 6], 2) a comprehensive model of information system security [7], 3) a secure software development life cycle, 4) threat modeling, 5) a software reengineering approach to developing secure code, and 6) example cases demonstrating that approach. The workshop was evaluated internally and externally, leading to identification of potential barriers preventing faculty from incorporating workshop material into their own curriculum. Each workshop component was individually assessed. Open-ended suggestions for workshop improvement were collected from participants.

- *Barbara Endicott-Popovsky, Ph.D., Director for the Center of Information Assurance and Cybersecurity, Research Associate Professor, Information School, U of Washington, Seattle, WA.*
- *Sam Chung, Ph.D., Associate Director of Cyber Physical Systems for the Center of Information Assurance and Cybersecurity, Endowed Professor of Information Systems and Security, Institute of Technology, U of Washington Tacoma, WA.*
- *Viatcheslav Popovsky, Ph.D., Affiliate Professor, Center for Ethics, HPRD Deparment, U of Idaho, Moscow, ID.*

We begin with an enumeration of the workshop components, and then summarize what we learned from faculty attendees.

II. SECURE CODE WORKSHOP CURRICULUM

The workshop was conducted over two days. Nine instructors from several schools participated. Each taught programming classes at either the beginning or intermediate levels. The workshop proceeded from abstract concepts to hands-on exercises. Curriculum elements, in order taught, are described subsequently:

A. Pedagogical Model

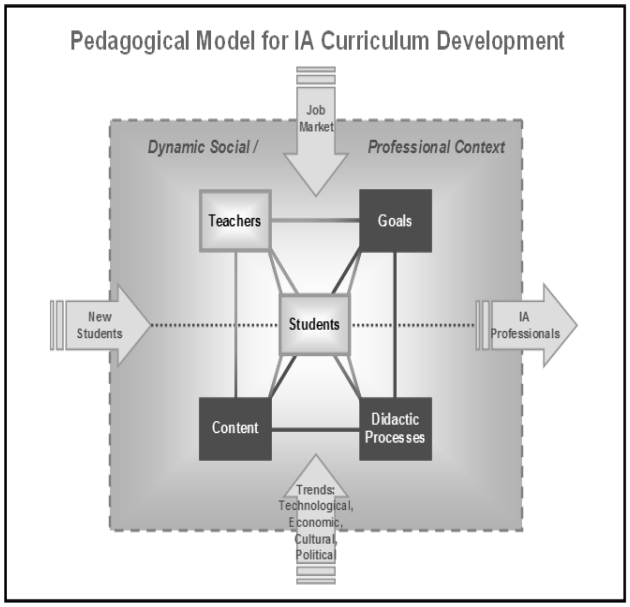

Figure 1: Pedagogical Model Underpinning Secure Code Curriculum Development

The authors have relied on the pedagogical model in Figure 1 for conceptualizing computer security and information assurance curricula [6]. The model has five elements: students, goals, content, the teacher and didactic processes. As a system, each component is influenced by the other. The more precisely the five components are defined, along with the connections among them, the more repeatable and predictable are the learning results [6].

Workshop attendees were encouraged to view their courses through this model, recognizing that as the student body demographics change from class to class, adjustments are needed in the other four elements. This is a particularly appropriate approach for returning adults whose backgrounds vary, cohort to cohort.

Applying the model, the following questions are asked: 1) How many elements comprise the subject? 2) At what level will each element be taught—reproductive vs. productive? 3) In what order should these elements be taught? 4) How much time should be spent on each element, and 5) how will students be tested?

B. Asset Projection Model (APM)

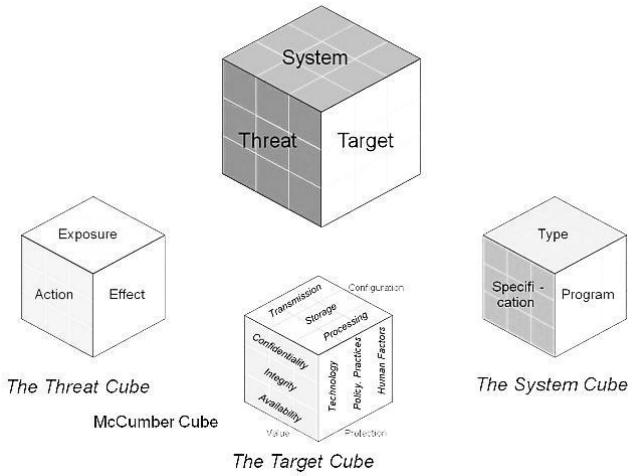

Figure 2: Asset Protection Model: An Extension of the McCumber Cube

To position secure coding within information assurance, workshop attendees were introduced to the Asset Protection Model (APM), Figure 2, which presents a systematic way to view the identification and protection of computer assets from three dimensions: 1) the system as a set of technical components, 2) the threat spectrum arrayed against it, and 3) the target characteristics a secure system must have. The model is described in depth in [6, 7].

The APM incorporates the Comprehensive Model of Information Systems Security (CMISS), commonly referred to as the McCumber Cube [8]. Presented in 1991, the McCumber Cube has remained useful to security practitioners over this extended period in spite of dramatic changes in technology due to its focus on information, rather than technology, along with a model structure of cognitive simplicity that allows human beings the ability to organize and reason about information at the proper level of abstraction.

C. Secure Development Lifecycle (SDL)

Several secure software development models were presented at the workshop. These included the Systems Security Engineering Capability Maturity Model (SSE-CMM) by the Software Engineering Institute at Carnegie Mellon University [9], the Software Assurance Maturity Model (SAMM) developed by Open Web Applications security Project (OWASP) [10], and Microsoft's Security Development Lifecycle (SDL) [11].

D. Threat Modeling Tool (TMT)

Threat modeling was demonstrated using Microsoft's Secure Development Lifecycle (SDL) Threat Modeling Tool (TMT) [12]. By inputting a Data Flow Diagram (DFD) into the TMT, it will indicate trust boundaries, identify potential threats according to the STRIDE model (**S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privileges), and suggest mitigations [13]. Figure 3 shows three trust boundaries in sample output from TMT version 3.1.4. This free download tool works on top of Visio.

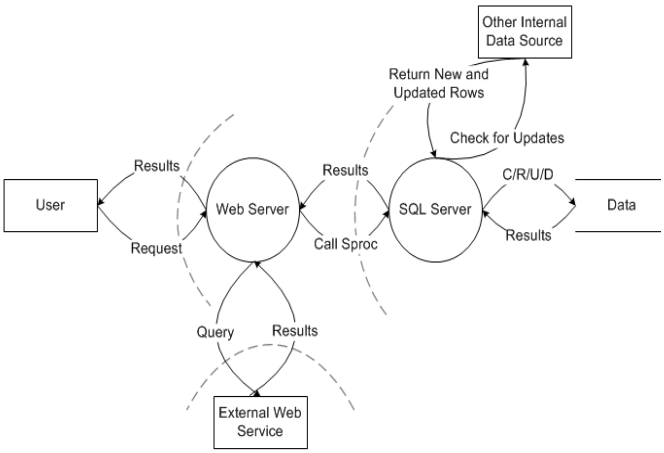

Figure 3: Sample Output from Microsoft's SDL TMT

E. Software Reengineering Based Secure Coding

We presented a specific software reengineering methodology we developed, Figure 4, which efficiently transforms non-secure source code examples from existing legacy assignments, into assignments resulting in secure code [14, 16]. It uses the re-documentation technique 5W1H Re-Doc [15] that employs UML diagrams to identify source code vulnerabilities, showing where security features can be inserted. We expected that teaching faculty participants this process would provide them a tool for easily converting their own legacy assignments into secure code assignments.

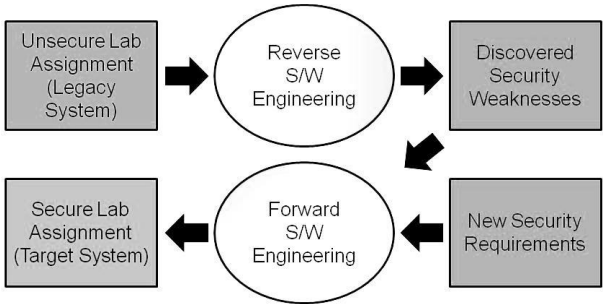

Figure 4: Software Reengineering Based Secure Coding Approach

F. Example Cases

Three example cases were used to demonstrate our software reengineering process. Video clips augmented presentations, walking viewers through the process in Figure 4, step-by-step, from analysis of vulnerable source code, using UML diagrams to identify code vulnerabilities, through forward engineering secure code from this analysis.

The first case demonstrated buffer overflow in a C program. The subsequent cases addressed database and Web site vulnerabilities, respectively, covering SQL Injection, and Cross Site Scripting (XSS). These cases were selected given frequency of occurrence.

III. Conducting the Workshop

The workshop targeted computer and information science instructors who teach beginning and intermediate programming or web development. Materials presented were made available to attendees in order to encourage use in the classroom. The reverse engineering process was described so that instructors could create their own example cases that could be shared later among attendees. A website was proposed for collecting and sharing these examples with a community of secure coding instructors.

Evaluators administered surveys at the end of both days of the workshop, in order to capture immediate feedback on discrete components. In addition, they observed interaction among attendees and presenters. Participant responses were identified by codes for anonymity. Responses were transcribed into a spreadsheet and analyzed using the IBM Statistical Package for the Social Sciences (SPSS).

IV. Key Findings

Key findings are summarized here and provided feedback for subsequent training events.

A. Background of Participants

Nine instructors participated. They either self-selected based on faculty announcements, or they were personally solicited by the organizers. As a result, participants had more than a passing interest in secure code. The workshop was a pilot for a series of workshops to be offered the following summer.

Attendees had worked in both industry and academia. Industry experience ranged from 3 to 30 years—averaging 14 years each. Instructor experience ranged from two quarters to 25 years—also averaging 14 years each. Six described their primary students as upper division, graduate, or professional returning adults. Two taught primarily first or second year undergraduate students. Both 4-year (7 faculty—5 from research schools, 2 from teaching-oriented schools) and 2-year institutions (2 faculty) were represented.

B. CS /IS Couse Curricula Represented

Participants reported teaching a wide variety of computer and information science topics, including operating systems, networking, system design, systems analysis, application development, computer hardware design, software engineering, data structures, web design and development, and compilers. Some also taught general classes such as computer and society, or technology and public policy.

Two taught entire courses in security, one actually taught a course in secure development and wanted to learn additional techniques. Six incorporated secure coding concepts in their courses. Three covered array overrun, SL and buffer injection, and input checking/data validation and three taught more complex topics.

Five observed that, as novices, students have difficulty grasping the implications of code vulnerabilities or inherent vulnerabilities in client-server architecture and networking, and therefore may not be candidates for many of the topics suggested in the workshop. In addition, even if students understood the concepts, they tended to ignore or underrate the importance of security and the need to be vigilant to creating vulnerabilities in their own code.

C. Goals for Learning Secure Coding Tools

Those who already taught some security concepts were asked what motivated them to do so. Some had been motivated by observing security challenges that continually occurred in student assignments, mentioning seeing repeated issues with memory management, input validation, constructor/destructor concerns and array bounds checking. Another identified being motivated by a desire to reduce vulnerabilities in deployed code; another wanted to influence students to adopt personal computer security measures.

At the end of the workshop, seven indicated intent to incorporate the workshop materials in some fashion. They were energized and enthused about some of our curriculum artifacts, in particular. They mentioned as especially useful: the National Vulnerability Database (NVD) http://nvd.nist.gov/ and Common Weakness Enumeration (CWE) http://nvd.nist.gov/cwe.cfm, code examples, the threat model, and the example cases.

Those inspired to teach security in the future were motivated by a desire to provide students with a set of secure code best practices, with one specifying wanting students to learn to "check boundary conditions of everything…" Other reasons given were the need for students to understand the risks associated with non-secure code, the need to develop a security attitude or culture, along with the need to raise student awareness. One indicated that a future desirable take-away would be a "Healthy fear of all aspects!"

Participants noted that they would add topics such as: secure code patterns; validation modules for students to insert in code; exercises where students try to break each other's code; methods to check input; multiple strategies for web security; exercises in Structured Query Language (SQL), and a demo in networking.

While seven indicated their curriculum would change as a result of the workshop, one said that theirs would not and the other gave a more noncommittal response.

V. ASSESSMENT

A. Workshop Assessment

Participants were asked to rate the workshop on a number of criteria, including how well they understood the material, how useful or relevant they found the material presented, how well it fulfilled their expectations, how well it met their needs, and how much they were looking forward to the second day's session. Figure 5 summarizes these results.

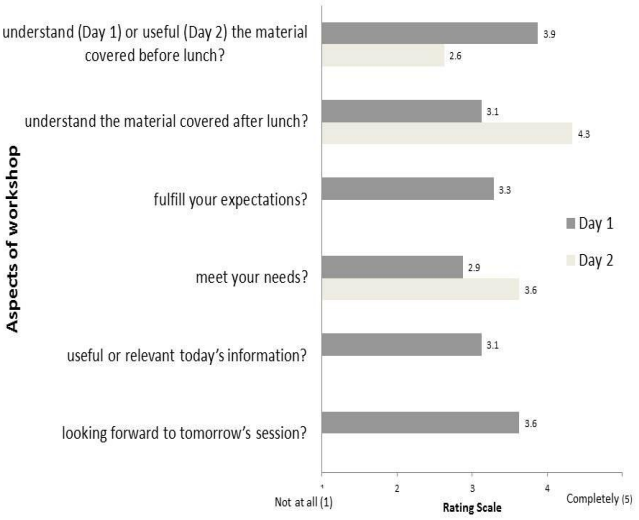

Figure 5. Workshop Assessment [17]

Seven of the participants asked for our classroom material and one indicated a need for ongoing support in order to implement them in their classes. As for what could be done better, they suggested:

- More examples of code (especially Java)
- More classroom materials
- Packaged validation functions for JavaScript/ PHP
- Packaged examples of XSS & SQL injection in JavaScript/PHP
- Resources made available on the web
- Additional, ongoing training

B. Barriers to Implementation

Figure 6 ranks the barriers viewed as preventing them from incorporating workshop material into their courses. Overall, participants viewed security topics as important and appropriate for their students, but they seemed uncertain about how to incorporate them, needing more supports in the form of resources, examples, and perhaps more training.

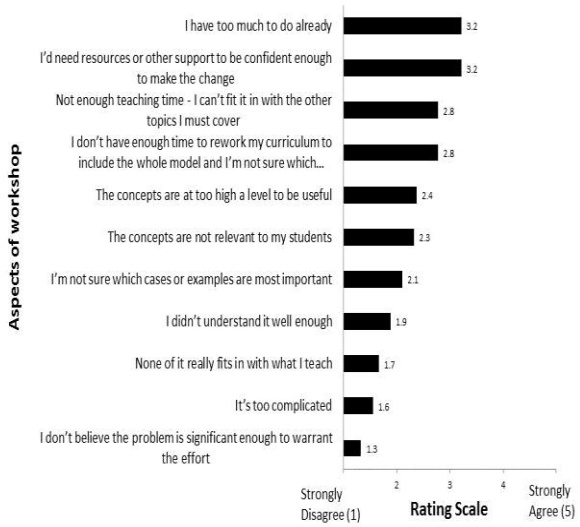

Figure 6. Potential Barriers to Implementation [17]

Faculty overload emerged as the biggest barrier to implementation. The three next most challenging barriers were the need for more resources, the lack of teaching time within the curriculum, and the lack of time to make the needed changes.

Two participants indicated that these challenges were great enough to prevent implementation. One thought that the concepts were at too high a level to be useful along with not having enough teaching time to fit the material in with other required topics.

The remaining potential barriers identified in the survey received less agreement. All participants disagreed with the statements: 1) "I don't believe the problem is significant enough to warrant the effort," (two-thirds of them disagreed strongly), and 2) "It's too complicated" (5 of the 9 disagreed strongly). Two thirds strongly disagreed with the statement "None of it really fits in with what I teach;" the other three were neutral.

C. Workshop Component –How Well Understood? How Relevant? How Likely to Implement?

Figure 7 provides insight into how the individual components of the workshop were regarded. In all cases, the components were better understood than they were perceived as useful. Further, the technical tools were ranked higher in value than the theoretical models. The threat modeling tool, the SDL, and the cases were the highest ranked and in that order. The software reengineering process was ranked in the middle, while the APM and pedagogical models were least valued.

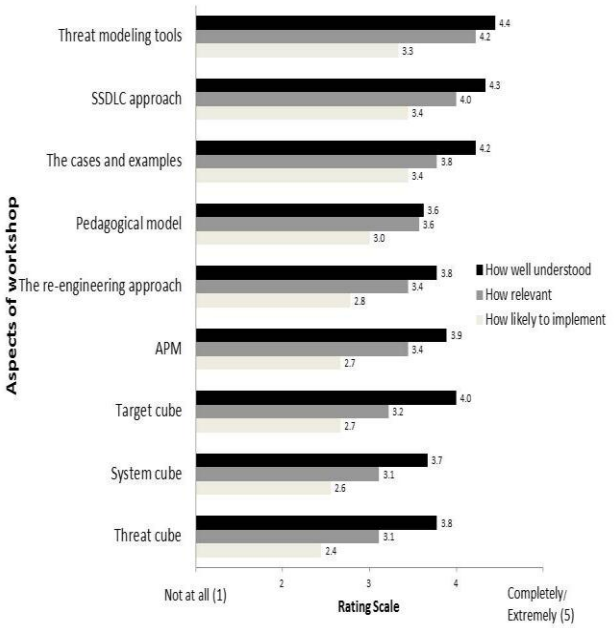

Figure 7. Attitudes & Plans Regarding Workshop Topics [17]

Participants were asked to rate how well they understood each workshop component, how relevant it was to the classes they teach, and how likely they were to implement it in their teaching within the next year. All questions used a scale from 1 (Not at all) to 5 (Completely). The responses are listed in descending order of relevance rating. These reinforce the comments and other data indicating that participants felt the least able to incorporate the theoretical models and the least convinced of their relevance to their students, and most able to apply the examples, approaches, and tools, seen as more relevant.

The threat modeling tool was seen as the most relevant topic covered, with seven of the nine selecting a response on the "relevant" side of the scale—four of them indicating "completely relevant." The other two selected the scale's midpoint. This was also the best understood workshop component with seven selecting a response above the midpoint (six of these selecting the top response), while the other two selected the scale's midpoint; however, only four indicated that they are likely to incorporate this tool into their curricula in the next year (two of these are extremely likely to do so) and three selected a response just below the midpoint.

Participants are as likely to implement the next two components:

- *The SDL approach*: eight understood it. Six found it relevant, and two rated its relevance below the midpoint. Five are likely to implement it, while three are unlikely.
- *The cases and examples*: seven understood them. Six found them relevant and the others selected the scale's midpoint. Four are likely to implement them, and four selected the midpoint.

Overall, the APM model and the associated target, system, and threat cubes were seen as least relevant to the courses taught by the participants. One individual reported that s/he did not understand the APM model at all and two said they understood it completely. One person said that it was not at all relevant to his/her classes and two said it was completely relevant. Two are not at all likely to incorporate it into their curriculum, and two are extremely likely to do so.

D. Final Participant Comments

Participants were generous with their final comments. When asked what components of the workshop we should continue to use in the future, participants were evenly divided between the practical and the theoretical, with three suggesting we keep the case examples and hands on exercises, and another three suggesting we keep the APM model.

When asked what to change, three asked for less theory. There were several suggestions regarding format: more opportunity to process the information through more discussion, more group work, and more hands-on exercises. One suggested adding a validation library for each programming language.

Participants were asked to reflect on why it has been hard to teach secure coding in their courses. Two themes emerged: not knowing what to teach; and not having time to do it. One person brought up the need to collaborate with other faculty members within his/her program.

In a follow up question, participants were asked whether it remains difficult content to teach after exposure in the workshop. Five indicated that it would not be as difficult to teach secure coding concepts, with two adding that the workshop moved their thinking forward on the subject. Two indicated that it would be difficult, still, but for different reasons. One needed more concrete examples to fully grasp the subject; another had a concern that security must be taught in context, suggesting that modules might be difficult to insert as is, depending on context, calling for a "deeper approach" to instruction in secure coding.

When asked for other comments, two individuals offered some additional feedback, suggesting ways to make the workshop more effective in the future. Both suggested making future workshops more product-oriented. This perspective was consistent with other comments made by the same individuals—that we could better meet the needs of future workshop attendees from computer science programs who are likely more inclined to learning from hands-on exercises.

VI. LESSONS LEARNED AND FUTURE WORK

Through the secure coding workshop, we confirmed that lack of time (either in the curriculum itself or in faculty time to develop new course materials or alter existing ones) hindered participants from injecting security topics into their own courses, although they recognized that security topics are important. Secondly, although participants could easily understand secure coding concepts—like SDL, the APM model, and threat modeling—and liked the hands-on practices, they strongly wanted more already prepared videos and assignment examples that demonstrate the transformation of non-secure to secure code.

This surprised us since we expected that being able to produce their own assignments from existing legacy assignments would engage them. We attributed this lack of interest in our process to lack of time to prepare their own materials, even if given an efficient process to convert already created materials. We did conclude from their responses that the software reengineering approach using UML modeling was a good method for teaching secure coding concepts, but primarily if the examples are presented in completed videos, instead of having the participants apply the method to their own legacy lab assignments to produce exercises themselves.

Based on the participants' responses, we concluded that the Microsoft TMT and SSDL were well-understood.; however, likeliness-to-implement was not as high, again given time constraints of the majority of participants. We concluded that preparing for the next workshop offering, we must create threat model examples and a variety of different cases in different languages so that they are ready to use by participants, as opposed to instructing them on how to build their own. Given the popularity of the TMT and the SDL, we plan to invite guest lecturers from industry who have used these tools to present how these tools have assisted them in improving the security of their products. We expect that this added emphasis may encourage instructors to use these tools in their classes.

Although they appeared to have a high degree of acceptance, the cases and examples developed for the workshop, in retrospect, could have been improved. Participants challenged whether the cases were sufficiently "real life." To improve their relevance, we will tie our cases more directly to the NVD and CWE. The APM model was seen as a reasonable way to position the subject of secure code within the information

assurance body of knowledge, although participants were vague about how to use the model in their classes. Since the model is a first draft and will be subject to additional reviews and iterations, this response is not surprising. Once the elements in the cubes are refined further, participants may see how the model could be useful within a secure code curriculum.

The tepid response to using the re-engineering approach was not entirely surprising, since many participants are not familiar with UML diagrams; however when presented with videos of a completed reengineering process, including narrative, the approach made more sense. In future presentations we need to show more clearly the integration between the re-engineering process and each case study. In the future, we plan to create instructor sets for each case that contain: power point presentations for lectures, demo videos that show use and misuse cases, demo videos that show potential attack vectors using UML diagrams, as well as how to reverse engineer the attacks using UML diagrams--relating the attacks to the NVD and CWE, and student assignments. By doing this, participants may see more readily how our re-engineering approach can be useful to reverse engineering non-secure legacy applications into more secure target applications.

Feedback from the workshop is the basis for major changes to the next iterations. Several more workshops are planned, this time for entire faculties who have expressed interest in this work. We expect that additional data from a larger population of faculty, both interested and not interested in secure coding, will give us further insight to improving workshop content and the uptake of secure coding concepts into computer and information science curricula. Evaluations will include collection of data after instructors teach their first classes, as a further measure of the effectiveness of these curricular assets.

The authors acknowledge that this study does not resolve the issue of thread model vs. the single course/track approach. It does provide for the first time, an efficient solution set for attempting the thread approach.

ACKNOWLEDGMENT
This research has been supported by the NSF (National Science Foundation) DUE (Division of Undergraduate Education) Federal Cyber Service: Scholarship for Service (SFS) under Grant No. 0912109.

REFERENCES

[1] Endicott-Popovsky, B.E., Frincke, D., V.M. Popovsky. (2005). *Secure Code: The Capstone Class in an IA Track*. Proceedings of the 9th Colloquium for Information Systems Security Education, Georgia Institute of Technology: Atlanta, GA, pp.100-108.

[2] Symantec's Vulnerability Trends http://www.symantec.com/business/threatreport/topic.jsp?id=vulnerability_trends&aid=total_number_of_vulnerabilities (Accessed on 3/2/2012)

[3] Perrone, L. F., Aburdene, M., and X. Meng. (2005). *Approaches to Undergraduate Instruction in Computer Security*, Proceedings of the American Society for Engineering Education Annual Conference & Exposition: Portland, OR.

[4] Chung, S. and Endicott-Popovsky, B. (2010). *Software reengineering based security teaching*. Proceedings of the 7th Annual International Conference on International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2010). Orlando, FL.

[5] Taylor, B. and S. Azadegan. (2008). *Moving Beyond Security Tracks: Integrating Security in CS0 And CS1*. Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, Portland, OR, p. 320-324.

[6] Simpson, J., Simpson, M., Endicott-Popovsky, B., and V. Popovsky. *Secure software education: A contextual model-based approach*. International Journal of Secure Software Engineering, 1(4), 35-61, October-December 2010.

[7] Simpson, J. and Endicott-Popovsky, B. (2010). *System security capability assessment model development and application*. In Proceedings of 20th Anniversary INCOSE International Symposium (INCOSE). Chicago, IL.

[8] McCumber, J. R. (1991). *Information systems security: A comprehensive model*. In Proc. 14-th NIST-NCSC National Computer Security Conference, Washington D.C., pages 328–337, October 1991.

[9] SSE-CMM (Systems Security Engineering-Capability Maturity Model) http://www.sse-cmm.org/index.html (Accessed on 6/27/2011)

[10] OWASP Software Assurance Maturity Model. https://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model (Accessed on 6/27/2011)

[11] Microsoft Security Development Lifecycle, http://www.microsoft.com/security/sdl/discover/default.aspx (Accessed on 6/27/2011).

[12] Microsoft's SDL Threat Modeling Tool, http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx (Accessed on 6/27/2011)

[13] Howard, M and D. LeBlanc. (2002). Writing Secure Code, (2nd ed.). Seattle: Microsoft Press.

[14] Hansel, L., Chung, S., Endicott-Popovsky, B. (2011). *Software Reengineering Approach to Teaching Secure Coding Practices*, the 15th Colloquium for Information Systems Security Education (CISSE 2011), June 13-15, 2011, Fairborn, Ohio.

[15] Chung, S., Won, D., Baeg, S. H., Park, S. (2009). *Service-Oriented Reverse Reengineering: 5W1H Model-Driven Re-Documentation and Candidate Services Identification*, In Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA'09) December 14-15, 2009, Taipei, Taiwan.

[16] Chung, S., and Endicott-Popovsky, B. (2010). *Software Reengineering Based Security Teaching*, International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2010). June 29 - July 2, 2010, Orlando, FL.

[17] Moore, E., Popovsky, V., Bai, Y., Taylor, C. (2010) Secure Coding Workshop: First Iteration Evaluation Report, Applied Inference, Seattle, WA.