Integrating Software Assurance Knowledge Into Conventional Curricula

Dr. Nancy R. Mead
Software Engineering Institute

Dr. Dan Shoemaker
University of Detroit Mercy

Jeffrey A. Ingalsbe
Ford Motor Co.

One of our challenges is deciding how best to address software assurance in university curricula. One approach is to incorporate software assurance knowledge areas into conventional computing curricula. In this article, we discuss the results of a comparison of the Common Body of Knowledge (CBK) for Secure Software Assurance with traditional computing disciplines. The comparison indicates that software engineering is probably the best fit for such knowledge areas, although there is overlap with other computing curricula as well.

Defects are not an option in today's world. Much of our national well-being depends on software. So the one thing that America's citizens should be able to expect is that that software will be free of bugs. Sadly, that is not the case. Instead, *commonly used software engineering practices permit dangerous defects that let attackers compromise millions of computers every year* . That happens because *commercial software engineering lacks the rigorous controls needed to (ensure defect free) products at acceptable cost* [1].

Most defects arise from program or design flaws, and they do not have to be actively exploited to be considered a threat [2, 3]. In fiscal terms, the exploitation of such defects costs the American economy an average of $60 billion dollars a year [4]. Worse, it is estimated that *in the future, the nation may face even more challenging problems as adversaries – both foreign and domestic – become increasingly sophisticated in their ability to insert malicious code into critical software systems* [3].

Given that situation, the most important concern of all might be that the exploitation of a software flaw in a basic infrastructure component such as power or communication could lead to a significant national disaster [5]. The Critical Infrastructure Taskforce sums up the likelihood of just such an event in the following:

> The nation's economy is increasingly dependent on cyberspace. This has introduced unknown interdependencies and single points of failure. A digital disaster strikes some enterprise every day, [and] infrastructure disruptions have cascading impacts, multiplying their cyber and physical effects. [5]

Predictions such as this are what motivated the National Strategy to Secure Cyberspace, which mandates the Department of Homeland Security (DHS) to do the following:

> ... promulgate best practices and methodologies that promote integrity, security, and reliability in software code development, including processes and procedures that diminish the possibilities of erroneous code, malicious code, or trap doors that could be introduced during development. [5]

Given the scope of that directive, one obvious solution is to ensure that secure software practices are embedded in workforce education, training, and development programs nationwide. The problem is that there is currently no authoritative point of reference to define what should be taught [3]. For that reason, in 2005 DHS created a working group to define a CBK for Secure Software Assurance <https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs/95.html>. The goal of the CBK is to itemize all of the activities that might be involved in producing secure code. DHS does not intend the CBK to be used as a general *standard, directive, or policy* [3]. Instead, its sole purpose is to catalog secure practices that might be appropriate to currently existing academic disciplines. Thus, the CBK is an inventory of potential *knowledge areas* within each contributing discipline.

The CBK assumes the following:

> ... software assurance is not a separate profession. What is not clear, however, is the precise relationship between the elements of the CBK and the curricula of each potentially relevant field. [6]

So, the challenge is to correctly integrate secure software assurance practices into each contributing discipline [3, 6].

Several disciplines could conceivably benefit from CBK, such as *software engineering, systems engineering, information systems security engineering, safety, security, testing, information assurance, and project management* [3]. Consequently, in order to ensure that the right content is taught in each, it is necessary to understand the proper relationship between the CBK and the curricula of each relevant discipline [6].

Finding Where the CBK Fits Into Current Curricula

The overall goal of the CBK is to *ensure adequate coverage of requisite knowledge areas in each contributing discipline* [3]. Accordingly, the working group sought to understand the exact relationship of CBK elements to each traditional curriculum. Once that relationship was better understood, it was felt that it should be possible to recommend the right way to incorporate CBK content into each of the established disciplines.

That comparison was materially aided by the fact that the sponsoring societies of the three most influential academic studies had just finished their own survey of curricular models for computing curricula. This was reported in "Computing Curricula 2005: The Overview Report" commonly called CC2005 [7]. CC2005 merges the recommendations for the content and focus of *computer engineering, computer science, information systems, information technology, and software engineering* curricula into a single authoritative summary, which is fully endorsed by the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE) Computer Society, and the Association for Information Systems.

CC2005 specifies 40 topic areas. These 40 topics represent the entire range of subject matter for all five major computing disciplines. The report specifically states the following:

> Each one of the five discipline-specific curricula represents the best judgment of the relevant professional, scientific, and educational associations and serves as a definition of what these degree programs should be and do. [7]

In addition to the 40 topic areas, which in effect capture all of the knowledge requirements for computing curricula

along with a ranking of their relative emphasis in each specific discipline, CC2005 also *summarizes the expectations for the student after graduation* [7]. This summary identifies 60 *competencies* that should be expected for each graduate. By referencing those identified competency outcomes, it is relatively easy to see the relationship between CBK knowledge elements and the CC2005 curricular requirements. It is also easier to see the places where there is a misalignment between the CBK and each discipline's curricular goals.

The CBK was mapped to the CC2005 recommendations for only three of the five disciplines. The two disciplines at opposite ends of the CC2005 continuum, *computer engineering* and *information technology*, were omitted because the former overlaps too much with electrical engineering and the latter overlaps too much with business.

Because these three curricula (computer science, information systems, and software engineering) have differing focuses, the content of CC2005 was examined one discipline at a time. First, a topic-by-topic analysis of *depth of coverage* was done. Depth of coverage was defined as *the quantity of material in the CBK that provides specific advice about how to execute a given activity in CC2005 in a more secure fashion.*

To obtain a metric, the assessment of *quantity of material* was based on a count of the textual references in the CBK that could be associated with each of the 40 topics. The assumption was that the more references to the topic in the CBK the greater the importance of integrating secure software assurance content into the teaching of that topic (see Table 1 for degree to which CC2005 Knowledge Areas are reflected in the CBK).

Knowledge Areas All Disciplines	Cites	Knowledge Areas All Disciplines	Cites
Integrative Programming (integrated)	9	Analysis of Requirements	1
Algorithms	1	Technical Requirements Analysis	274
Complexity	6	Engineering Economics for Software	1
Architecture	150	Software Modeling and Analysis	2
Operating Systems Principles and Design	5	Software Design	255
Operating Systems Configuration and Use	5	Software V&V	401
Platform Technologies	3	Software Evolution (Maintenance)	438
Theory of Programming Languages	10	Software Process	296
Human-Computer Interaction (HCI)	5	Software Quality	163
Graphics and Visualization	1		
Information Management (DB) Practice	1	**Non-Computing Topics**	**Cites**
Legal/Professional/Ethics/Society	93	Risk Management	86
Information Systems Development	7	Project Management	156

Table 1: *Degree to Which CC 2005 Knowledge Areas Are Reflected in the CBK*

What Does This Mean?

The following eight CC2005 topic areas had a *significant* degree of coverage in the CBK (greater than 100 references): 1) requirements, 2) architecture, 3) design, 4) verification and validation (V&V), 5) evolution (e.g., maintenance), 6) processes, 7) quality, and 8) information systems project management. The following three CC2005 topic areas had *moderate* coverage in the CBK (less than 100 but more than 10): 1) legal/professional/ethics/society, 2) risk management, and 3) theory of programming languages.

This mapping shows that the main focus of the CBK is on generic software *work* rather than on the specific curricular aspects that characterize the study itself, such as algorithms (for computer science), or information systems management (for information systems). That indicates that CBK content would be best integrated into the places where the practical elements of the life cycle are introduced, such as a software design project course.

Fit Between the CBK and Desired Outcomes for the Profession

There were six priorities in CC2005. These range from *highest possible expectations* through *highest expectations* to *moderate expectations*, *low expectations*, *little expectations*, and *no expectations*. One of the more interesting aspects of CC2005 is the 60 expected competencies. Because there is a difference in focus for each discipline, there is a difference in what should be expected for each of them. For instance, there is a different set of presumed competencies for a computer scientist than for a software engineer.

The 60 expected competencies were taken directly from the 40 learning topics. Each competency was examined to determine which of the 40 topics could be assigned to it. For instance, if the competency was to *design a user-friendly interface*, there are 255 references in the CBK to *design*, and five references in the CBK to *human/computer interfaces*. So the number of CBK references for this outcome was assigned as 260 (Tables 2-4, pages 18-19).

For *computer science* there is a weak match between the CBK and *the highest possible competency expectations* in that only one of the eight outcomes (12.5 percent) had any degree of coverage. There is a slightly better match for the high expectations category, three of 10 (30 percent). However, there is an excellent match with moderate expectations, nine of 12 (75 percent).

For *information systems* there is a reasonable match between the CBK and the *highest possible competency expectations* in that nine of the 22 competencies (40.9 percent) specified for that discipline are covered. There is no match for the high expectations category. However, there is a good match with moderate expectations, five of nine (55.5 percent).

For *software engineering* there is a strong match between the CBK and the *highest possible set of competency expectations* in that four of the seven outcomes (57.1 percent) are covered. There is reasonable match for the high expectations category in that three of 12 competencies are covered (25 percent). There is also a good match with moderate expectations in that five of nine areas are covered (55.5 percent).

Integrating the CBK into the World of Practical Education

One of the main inferences that can be drawn from this comparison is that the current CBK is less focused on theory than it is on application of the knowledge in practice. In essence, the results demonstrate that the CBK is built around and encapsulates knowledge about practical processes that are universally applicable to securing software rather than on discipline-specific concepts, theories, or activities.

This is best illustrated by the matches themselves. *Outcomes such as solve programming*

Computer Science		
Highest Possible Expectation (5)	**Depth of Coverage**	**Conclusion**
Solve programming problems (algorithms)	Analysis (274), Design (255)	strong
High Expectation (4)	**Depth of Coverage**	**Conclusion**
Do large-scale programming (programming)	Analysis (274), Design (255), Architecture (150)	strong
Develop new software systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
Create a software user interface (HCI)	HCI (5)	weak
Moderate Expectation (3)	**Depth of Coverage**	**Conclusion**
Create safety-critical systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
	Process (296), V&V (401), PM (156)	
Design information systems (IS)	Analysis (274), Design (255), Architecture (150)	strong
Implement information systems (IS)	Process (296), V&V (401), PM (156)	strong
Maintain and modify information systems (IS)	Evolution (438)	strong
Install/upgrade computers (planning)	Process (296), V&V (401), PM (156)	strong
Install/upgrade computer software (planning)	Process (296), V&V (401), PM (156)	strong
Design network configuration (networks)	Analysis (274), Design (255), Architecture (150)	strong
Manage computer networks (networks)	Evolution (438)	strong
Implement mobile computing system (networks)	Analysis (274), Design (255), Architecture (150)	strong

Table 2: *Match Between CBK and CC2005 Expected Competencies for Computer Science*

problems, *do large scale programming*, and *design and develop new software and/or IS are high priorities in all of the disciplines*. At the same time, each of these also has a significant degree of coverage in the CBK.

High-priority items in each of these disciplines that were not good matches with the CBK tended to be such competencies as *prove theoretical results* (computer science), *develop proof-of-concept programs* (computer science), *select database products* (information systems), *use spreadsheet features well* (information systems), *do small scale programming* (software engineering), and *produce graphics or game software* (software engineering).

Others, such as *create a software user interface*, were a mixed bag, with a good match to design but a poor match to HCI.

So, while these competencies might be individually important to their specific disciplines, they are not essential elements of

Table 3: *Match Between CBK and CC2005 Expected Competencies for Information Systems*

Information Systems/Information Technology		
Highest Possible Expectation (5)	**Depth of Coverage**	**Conclusion**
Create a software user interface (IS)	HCI (5)	weak
Define information system requirements (IS)	IS Dev. (7), Bus. Req. (1), Analysis (274)	strong
Design information systems (IS)	Design (255), Modeling (5), Architecture (150)	strong
Maintain and modify information systems (IS)	Evolution (438)	strong
Model and design a database (DB)	DB (1), Design (255)	strong
Manage databases (DB)	Evolution (438)	strong
Develop corporate information plan (planning)	Process (296)	strong
Develop computer resource plan (planning)	PM (156)	strong
Schedule/budget resource upgrades (planning)	PM (156)	strong
Develop business solutions (integration)	Bus. Req (1), Modeling (5), Design (255)	strong
High Expectation (4)	**Depth of Coverage**	**Conclusion**
None	n/a	
Moderate Expectation (3)	**Depth of Coverage**	**Conclusion**
Develop new software systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
Install/upgrade computer software (planning)	PM (155)	strong
Manage computer networks (networks)	Evolution (238)	strong
Manage communication resources (networks)	PM (155), Economics (1)	strong

Software Engineering		
Highest Possible Expectation (5)	**Depth of Coverage**	**Conclusion**
Do large-scale programming (programming)	Analysis (274), Design (255), Architecture 150)	strong
Develop new software systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
Create safety-critical systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
	Process (296), V&V (401), PM (156)	strong
Manage safety-critical projects (programming)	V&V (401), PM (156), Evolution (438)	strong
High Expectation (4)	**Depth of Coverage**	**Conclusion**
Develop new software systems (programming)	Analysis (274), Design (255), Architecture (150)	strong
Create a software user interface (HCI)	HCI (5)	weak
Define information system requirements (IS)	IS Development (7), Bus. Req. (1), Analysis (274)	strong
Moderate Expectation (3)	**Depth of Coverage**	**Conclusion**
Design a human-friendly device (HCI)	HCI (5), Design (255)	strong
Design information systems (IS)	Design (255), Modeling (5), Architecture (150)	strong
Maintain and modify information systems (IS)	Evolution (438)	strong
Install/upgrade computers (planning)	Process (296), V&V (401), PM (156)	strong
Install/upgrade computer software (planning)	Process (296), V&V (401), PM (156)	strong
Manage computer networks (networks)	Evolution (438)	strong
Implement mobile computing system (networks)	Analysis (274), Design (255), Architecture (150)	strong

Table 4: *Match Between CBK and CC2005 Expected Competencies for Software Engineering*

secure software assurance as defined by the CBK. In view of that finding, it would appear to be easier to introduce CBK content into curricula that is focused on teaching pragmatic software processes and methods. And given its historic involvement with those areas, the discipline of software engineering might be the place to start.

Therefore, one additional suggestion might be that a similar study should be done based strictly on Software Engineering 2004, "Curricular Guidelines for Undergraduate Programs in Software Engineering," particularly Table 1: Software Engineering Education Knowledge Elements [8]. That itemizes a set of *knowledge areas and knowledge units* that are similar in focus and purpose to the 40 knowledge areas contained in CC2005. Thus, it should be possible to better understand the actual relationship between standard software engineering curricular content and the contents of the CBK through that comparison.

There is another distinct observation arising out of this study. Although the *moderate expectations* category does not reflect priority areas, it is overwhelmingly the best aligned category for *each* discipline. What that might indicate is that, although secure software assurance is a legitimate area of study for all of these fields, it is not the highest priority in any of them. In terms of disciplinary implementations, the practitioner orientation and the fact that security content is not the point of these fields indicates that courses that cover practical lifecycle functions might be the place to introduce secure software assurance content within any given discipline.

As a final note, the measurement process used in this study (e.g., a raw count) is inherently less accurate than expert contextual analysis of the meaning of each knowledge element. Therefore, a more rigorous comparison should be undertaken to better characterize the functional relationship between the items in the CBK and the various curricular standards. This would be particularly justified for the study of software engineering curricular content mentioned above. Once a means of comparison that everybody can agree on is used, it should be relatively simple to work out the nuts-and-bolts of specific implementations within each individual program.◆

References

1. President's Information Technology Advisory Committee. Cybersecurity: A Crisis of Prioritization. Arlington: Executive Office of the President, National Coordination Office for Information Technology Research and Development, 2005.
2. Jones, Capers. Software Quality in 2005: A Survey of the State of the Art. Marlborough: Software Productivity Research, 2005.
3. Redwine, Samuel T., Ed. Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire and Sustain Secure Software, Version 1.1. Washington: U.S. DHS, 2006.
4. Newman, Michael. Software Errors Cost U.S. Economy $59.5 Billion Annually. Gaithersburg: National Institute of Standards and Technology (NIST), 2002.
5. Clark, Richard A., and Howard A. Schmidt. A National Strategy to Secure Cyberspace. Washington: The President's Critical Infrastructure Protection Board, 2002 <www.us-cert.gov/reading_room/cyberspace_strategy.pdf.>.
6. Shoemaker, D., A. Drommi, J. Ingalsbe, and N.R. Mead. "A Comparison of the Software Assurance Common Body of Knowledge to Common Curricular Standards." Dublin: 20th Conference on Software Engineering Education and Training, 2007.
7. Joint Taskforce for Computing Curricula. Computing Curricula 2005: The Overview Report. ACM/AIS/IEEE, 2005.
8. Joint Taskforce for Computing Curricula. Software Engineering 2004, Curricular Guidelines for Undergraduate Programs in Software Engineering. ACM/IEEE, 2004.

About the Authors

Nancy R. Mead, Ph.D., is a senior member of the technical staff in the Networked Systems Survivability Program at the SEI. She is also a faculty member at Carnegie Mellon University. Mead's research interests are in the areas of information security, software requirements engineering, and software architectures. She is a Fellow of the IEEE and the IEEE Computer Society and is also a member of the Association for Computing Machinery. Mead received her doctorate in mathematics from the Polytechnic Institute of New York, and received bachelor's and master's degrees in mathematics from New York University.

SEI
4500 5th AVE
Pittsburgh, PA 15213
E-mail: nrm@sei.cmu.edu

Dan Shoemaker, Ph.D., is the director of the Centre for Assurance Studies. He has been professor and chair of computer and information systems at the University of Detroit Mercy for 24 years, and co-authored the textbook, "Information Assurance for the Enterprise." His research interests are in the areas of secure software assurance, information assurance and enterprise security architectures, and information technology governance and control. Shoemaker has both a bachelor's and a doctorate degree from the University of Michigan, and master's degrees from Eastern Michigan University.

Computer and Information Systems – College of Business Administration
University of Detroit Mercy
Detroit, MI 48221
Phone: (313) 993-1202
E-mail: shoemadp@udmercy.edu

Jeffrey A. Ingalsbe is a senior security and controls engineer with Ford Motor Company where he is involved in information security solutions for the enterprise, threat modeling efforts, and strategic security research. He has a bachelor's degree in electrical engineering and a master of science degree in computer information systems from Michigan Technological University and the University of Detroit Mercy, respectively. He is currently working on a doctorate in software engineering at Oakland University. Ingalsbe serves as an expert industry panelist on two national working groups within the DHS's Cybersecurity Division.

17475 Federal DR
STE 800-D04
Allen Park, MI 48101
Phone: (313) 390-9278
E-mail: jingalsb@ford.com