Proceedings of the 10th Colloquium for Information Systems Security Education
University of Maryland, University College
Adelphi, MD June 5-8, 2006

The Software Assurance CBK and University Curricula

Matt Bishop and Sophie Engle, *University of California at Davis*

Abstract – The recently proposed Secure Software Assurance Common Body of Knowledge is a first effort at collecting information about security-enhanced programming and systems development. One of its stated goals is to drive curriculum development in academic institutions. This paper analyzes the SwACBK's usefulness in programs for advanced undergraduate and graduate education, and offers suggestions for strengthening it.

Index terms – secure software, common body of knowledge

I. Introduction

The poor state of software today leads to non-secure, unreliable systems. The cause is complex. Some industry leaders blame academic institutions for not teaching students how to program securely, to prevent these problems. Some academics point to industry's drive to get products to market and then retrofit proper security into the product—a notoriously ineffective approach. Responsibility is difficult to assess, and ultimately meaningless. The question is what to do about the problem.

First, we need to define "the problem" precisely. It actually has three parts:

1. Teaching programmers the principles and techniques of secure software assurance design and implementation;
2. Giving programmers the freedom to use that knowledge and skill; and
3. Selecting, configuring and using that software properly.

This paper deals with the first part in the context of academic education. The goal of an academic education is to teach students not only how to do something, but also why one does it that way, and to provide a basis for them to question, extend, and ultimately improve or create new methods. For this type of education, the emphasis is on principles and understanding. Practice and technique play supporting roles. This enables the student to apply what she has learned to situations not presented or discussed in class, rather than being unable to extend the application of the technologies she has been taught.

The precise principles and techniques to teach are a subject of some debate. There is no "Common Body of Knowledge" that describes what a programmer should know to be considered competent in developing secure software. In 2004, the Department of Homeland Security and the Department of Defense began an effort to develop a common body of knowledge for secure software assurance. One of the goals of the effort is "to provide an inclusive list of the knowledge needed to acquire, develop, and sustain secure software."[1] An ancillary goal is to "help ... academia target [its] education and training curricula."[2] The "Secure Software Assurance Common Body of Knowledge" (SwACBK) [1] came from this effort.

The goal of this paper is to suggest changes to the SWACBK that will make it more useful as a basis for curriculum development in academia. Specifically, we argue that the SwACBK should be restructured to emphasize principles, that it should use a more comprehensive framework from which material at various levels of abstraction may be derived, and that it should include more seminal references.

The core problem with the SwACBK is that it is aimed at the need of government to procure secure software. In itself, this is not a problem. But generalizing from that goal to providing a basis for a sound academic curriculum assumes that the purpose of the curriculum is to educate students on how to write secure programs for the government. The two overlap, but have very different foci. We begin here, by reviewing the nature of teaching at universities.

II. Background

A. Common Body of Knowledge

A "common body of knowledge" is a collection of information and a framework that provides a basis for understanding terms and concepts in a particular knowledge area. It also provides a set of basic information that people who work in that knowledge area are expected to know.

Authors' affiliation: Department of Computer Science, University of California at Davis, One Shields Ave., Davis, CA 95616-8562 USA

[1] [1], p. x
[2] [1], p. x.

The content of a common body of knowledge depends in part upon the use to which the information will be put. The SwACBK's definition is the knowledge needed to perform "the engineering activities or aspects of activities that are relevant to achieving secure software"[3]. Hence, its goal is to provide a framework for the information deemed necessary to perform those activities. That it is a common body of knowledge also means that one goal is to provide a foundation upon which an academic curriculum can be built, thereby influencing the content of that curriculum.

B. Academic Curriculum

The purpose of an academic education is to teach students how to synthesize information, principles, concepts, and other materials to be able to apply it to novel situations. Teaching for this purpose focuses on principles and concepts, illustrating them by using methodologies and technologies. Two approaches are common.

The first approach is to begin with the concepts and principles, and illustrate them with examples. Teaching the Flaw Hypothesis Methodology (FHM) using this approach would begin by describing the technique. Then the instructor would illustrate how to apply the methodology to some example system. The system need not be real, although if it were a system that students were familiar with, the instructor could lead the class in a simple exercise.

The second approach is to begin with examples, and from them derive the broader concepts and principles. To teach the Principle of Least Privilege, for example, an instructor might present several instances where too many privileges caused breaches of security. From these cases, the conclusion follows that the level of privilege should be a minimum—stating the principle being taught.

Both methods emphasize principles and concepts; they differ only in how the students are led to those. Training would focus specifically on how to develop secure software for particular environments. Training curricula are therefore fundamentally different than academic curricula, although they may contain some common material. Two examples illustrate this claim.

The Computer Science Accreditation Board (CSAB) accredits many academic programs. The key points of a curriculum that will succeed in being accredited include Curriculum Standard IV-7, that the "[t]heoretical foundations, problem analysis, and solution design must be stressed within the program's core materials"[4]. Proposed criteria amplify this; for example, Criterion 1(b) states that an accredited program is to enable students to "apply design and development principles in the construction of software systems of varying complexity"[5]. Throughout, the criteria stress students understanding principles and how to apply them.

Similar requirements exist on university campuses. The University of California campuses are highly respected institutions of graduate and undergraduate education. Each campus has a division of the systemwide Academic Senate, composed of all ladder faculty at that institution. The Committee on Courses of Instruction sets regulations for approval of courses. The Davis Division's Committee on Courses of Instruction's policies and procedures are typical of the nine campuses. Item II.A.1 states that a "University course should present an integrated body of knowledge, with primary emphasis upon elucidation of principles and theories rather than upon development of skills and techniques"[6]. The requirement for graduate courses, III.B.1.c, is stated more specifically: they "have a strong foundation in the theory, methods and principles used in research" and "focus on understanding and assessing the current state of knowledge, on research [...], and on methodology"[7]. Again, the emphasis is on principles and concepts at all levels of courses.

As a counterpoint, checklists are typically used in training, with some exceptions [4]. In academic curricula, the focus is on how to *derive* checklists, taking into account the relevant principles, the goals of the checklists, and the environment in which the checklists are to be used. The designers of the checklist must understand the theory and principles underlying the actions on the checklist, even if the users do not.

C. Security-Enhanced Systems

The terms "secure software" and "secure systems" are misleading, because they imply absolutes. Strictly speaking, "secure" systems are systems that satisfy two properties:

1. The set of security requirements that the system is to satisfy is complete; and
2. The system is developed, deployed, and operated in a manner that provides sufficient assurance to assert that the system satisfies the security requirements.

Technologies to develop these systems are called *high assurance technologies* and use a multitude of methods. Unfortunately, these methods require expertise, time, and money, and so are not used often enough. Instead,

[3] [1], p. viii.
[4] [2], p. 3.
[5] [2], p. 17.
[6] [3], http://academicsenate.ucdavis.edu/ra/policy.htm#lvl&emph.
[7] [3], http://academicsenate.ucdavis.edu/ra/policy.htm#Course.

developers use methods that provide lower assurance. Given the state of most commercial systems and software, the gap between standard methods used in the industry, and high assurance methods, is painfully obvious.

The content of the SwACBK attempts to address the gap between industry standard methods and high assurance methods by providing a framework to teach students and practitioners about ways to develop security requirements and increase assurance. The SwACBK is *not* aimed at high assurance; it is aimed at assurance higher than most software and systems currently provide.

For brevity, we follow the standard practice of calling software "secure" if it is developed with the two properties above. The more rigorous the derivation of the security requirements, and the more convincing the evidence of assurance, the more secure the software is.

III. 3. Problems with the SwACBK

The SwACBK has several problems that inhibit its use in driving academic curriculum.

A. Secure Software in General

A basic aspect of security is interdependence. Software can be secure (for whatever definition of "secure" is appropriate) only when the system it runs on is secure, because if an attacker can disrupt the system resources on which the software relies, the software will fail. Similarly, secure software installed incorrectly may not be secure, because the installation process may use vulnerable passwords or cryptographic keys, or rely on flawed system components. Systems protected by flawed procedures are not secure, because an attacker can subvert those procedures; consider the success of social engineering, for example.

The SwACBK deals with this problem in §5.2.5, which states that "[s]ecurity inspired requirements on nature and attributes of computing hardware, infrastructure, or other externally available services must be explicitly recorded as requirements or assumptions and assured."[8] The problem is that, in chapter 8 ("Secure Software Verification, Validation, and Evaluation"), no discussion of these requirements and assumptions is given, leading the reader to believe that, once validated to be secure, the software may be moved to other environments safely. Assurance relies specifically on assumptions made, and assurance evidence rests on them. Software can never be "secure" but can only be secure with respect to a set of assumptions. The SwACBK fails to make this point effectively. Developers of curricula who rely on the guidance of the current SwACBK may well miss this point, and produce an inferior curriculum as a result.

B. Orientation

A common body of knowledge must address many realms, because security affects all of government, industry, and academia. The SwACBK developers understood this, and included members of each of those communities. But some sections focus on the needs and problems of government to such a degree that the environments of academia and industry are overlooked.

Consider for example §2.2, "Dangerous Effects." This section provides background so that readers will understand the threats and risks underlying software and system security. But all the examples for the section focus on attacks upon government agencies and national security, leaving the impression that the threat to software security is primarily a national security threat. Industry is mentioned only once, in a comment that large organizations such as the Department of Defense and Microsoft are probed often. Risks to academic institutions are not mentioned at all.

When motivating the need for a generally useful common body of knowledge, the goal is to convince readers that the work transcends a specific type of organization, and can be applied to a variety of institutions. This section convinces the reader of the grave threat to national security that poor software security poses, but does not explain why software developers working for private industry or academic institutions should be concerned.

A second example is in §7.2.1. The list of sources for information on vulnerabilities, exploits, and patches includes 2 government sites and 1 industry site. Well-known non-governmental sites such as Security Focus, the Open Source Vulnerability Database, and ISS' X-Force are omitted. As many of these sites provide more information than the government sites listed, their omission is surprising.

A guide to an academic curriculum needs greater breadth, because students may go into industry, or may become teachers. A curriculum must focus on the goal of the lessons, which are to teach generally applicable principles and concepts—and how to use them in a variety of environments, including government.

C. Classifications

In many places, the SwACBK presents simple classifications of ideas, or tools, or methodologies, in order to structure the discussion. But the classifications

[8] [1], p. 60.

are not rigorously derived from some stated basis, and hence are (at best) confusing and (at worst) misleading.

Two examples will show the nature of the problem. In §6.7, "Architectures for Security," the SwACBK provides a set of architectural styles or style elements for security-aware architectures. Listed are reference monitors, "layered", "system high", and "filters, guardians, and firewalls" to provide compartmentalization. The problem is that "layered" refers to a type of architecture in which security services are built (layered) on top of more basic security services. Underlying these services are reference monitors that control access to resources. Filters, guardians, and firewalls can be examples of reference monitors, or security mechanisms built upon reference monitors. System high refers to a security level for the most secure data. The elements of the list are not organized in a cohesive manner, nor is there any guidance about the relationship of elements in the list. A more useful list would begin with the reference monitor as the architectural underpinning of the elements of the list. As processes either run in isolation or communicate with other processes, the list would next enumerate mechanisms to do either. Above that comes how the processes should be organized: as multiple independent levels of security, multiple single levels of security, and so forth. Then might come the ways the mechanisms handle (attempted) violations: tolerate the intrusion, isolate the attacker, turn on deception, and so forth. The classification of the style and elements does not matter; that there be a structure or organization of the elements and styles within the classification does.

A second example arises in §2.4, "Method for Attacks," which tries to enumerate the different types of attacks. The document splits attacks into attacks against the operating system, against software, and against physical attacks. But this classification scheme is not exhaustive. For example, a covert channel may disclose information, but does not appear to fall into any of these categories. Further, it is not clear why these categories were chosen. What does this particular classification lead to? Why not use a model such as Cohen's attack classification [5], which has examples and detailed descriptions?

The lack of rigor in classifications makes constructing a curriculum that follows those classifications difficult, because the curriculum will not provide the correct relationships between the entities so classified. More importantly, there is little assurance that the classifications are complete (as is the case for the different types of attacks). Hence, a discussion following the SwACBK may omit some critical parts of the mechanisms under discussion.

Worse is the failure to build a framework upon which the classifications rest. The current schemes do not organize information well, something critical for people who are not entirely familiar with the field.

D. Basis and Depth

The SwACBK's §3, "Fundamental Concepts and Principles," presents a list of principles developed by Saltzer and Schroeder [6], with some additions. Two such additions (§3.3.11, "Defense in Depth" and §3.3.12, "Analyzability") are actually different views of existing principles (§3.3.6, "Separation of Privilege" and §3.3.3, "Economy of Mechanism", respectively). The rest of the chapter focuses on concepts such as reliability, assurance, and security.

The discussion of models mentions the Bell-LaPadula model (§3.5.1), but presents no integrity models. The lack of discussion of the Clark-Wilson Integrity Model is especially glaring,[9] as it combines concepts of assurance, integrity, and auditability, and reflects widespread practices in industries such as the financial community.

In the discussion of malicious logic ("malware"), computer viruses and worms are discussed, but their common ancestor, the Trojan horse, is mentioned only once,[10] and in the context of a "back door." The definition provided is that of a back door, and the generality of the Trojan horse is overlooked.

The importance of reference monitors and validation mechanisms to assurance, and hence to secure software and systems, is critical; yet reference monitors are mentioned only four times, and very briefly. Similarly, that proofs in assurance do not eliminate the need for testing[11] deserves an explanation, or discussion, because the reason that the statement is correct is subtle (and deals with the difference between abstraction and implementation).

As a final example, the SwACBK does not discuss the trade-off between dynamic analysis and static analysis. What are the benefits of each, and when is the use of each appropriate? The SwACBK cannot of course discuss this in depth, but it can (and should) mention that such a trade-off exists, and compare the two techniques in that light.

That the SwACBK does not emphasize principles enough produces most of the problems identified in this section. Beyond depth of understanding, principles provide an organization for ideas and methodologies built upon those principles. A curriculum founded upon such an organization will provide a more coherent basis for

[9] The paper by Clark and Wilson is referenced in section 3.10, Recommended Reading, and once in the Bibliography.
[10] [1], p. 17
[11] [1], p. 137

educating students, leading to a deeper understanding of not only how things work, but why they work, and how to approach new problems.

E. Motivation

Discussing *why* a particular issue is significant is often as important as discussing the problem itself, because it provides context. The SwACBK provides motivation for why software security needs to be addressed, and makes a good case for having a common body of knowledge. But it often omits the motivation for individual facets of software assurance; the result is that the reader has no sense of the importance of those aspects of software assurance.

As an example, consider §9.5, which discusses static analysis. The opening sentence states that static analysis techniques "are conservative, making worst case assumptions to ensure the soundness of the analysis."[12] The SwACBK does not explain this comment further. The problem is that the statement is true for a specific set of assumptions, namely that it is better for a static analysis technique to report security vulnerabilities erroneously (false positives) than fail to report a security vulnerability (false negative). Most analyzers (and techniques) can be configured to do the reverse, in which case the statement is incorrect. Motivating the statement by saying that false negatives are generally considered worse than false positives would provide the perspective that the reader needs to evaluate the comment.

F. References

Part of the goal of a common body of knowledge suitable for guiding curriculum is to make available references to seminal material, so faculty and students know where to look for more detailed expositions or historical origins. The SwACBK provides many references to contemporary material. In §3.8.1,[13] it mentions a collection of seminal papers. But with only one exception, the SwACBK does not identify any seminal works that discuss the concepts.

The discussion of reference monitors is a good example. The citation for the term is a book published in 2003 [7]. But James P. Anderson introduced the concept of a reference monitor in a seminal study in 1974 [8], and the term has been widely used ever since. While the book referenced explains the term, the lack of the original reference deprives the reader of any hint of the lineage of the term, or of how widely it is used.

The same study introduced the concept, and term, "Trojan horse" to computer security. As mentioned above, the SwACBK mentions "trojans" in the context of malware[14] but it gives no reference to the source document (or to any other document, for that matter). The only reference for malware in general is to a book written in 2005 [9][15]; again, the nuances of that term, and of the research underlying the results presented in the SwACBK, are never alluded to.

The SwACBK does refer to one seminal paper that discusses principles of secure design [6]. However, the only other references from before 1985 are the RISOS study of vulnerabilities (1976) [10], a book on anthropology (1978) [11:9], a book on software configuration management (1980) [12], a paper on designing for testability (1982) [13], a paper on obtaining agreement to change organizations (1983) [14], and Thompson's Turing Award lecture (1984) [15]. References for the Bell-LaPadula model, the Chinese Wall model, role-based access control, and originator-controlled access control are omitted, even though the models are named in the text[16].

G. Use in Higher Education Curriculum

The SwACBK makes several recommendations for incorporating the material it covers into a program of study in higher education. Because it distinguishes between undergraduate and graduate education, we focus on these separately.

The SwACBK suggests that either separate courses cover the knowledge area, possibly in conjunction with standard software engineering courses, or that existing courses be augmented with the material in the SwACBK. The biggest problem is that practicing the material requires reinforcement, so portions of it—especially the analysis and testing—must be embedded in the grading of programming assignments. Otherwise, like the good programming style taught in introductory programming classes, assigned programs will focus on whether the program "works" and not on assurance attributes such as robustness and security. So, whichever approach is adopted, the assignments given in other courses must also reinforce the material, by the grading of assignments if in no other way. The SwACBK should emphasize this last point.

The SwACBK suggests that graduate programs use the "guidance for training entry-level practitioners"[17] to address incoming (graduate) students. The guidance for training entry-level practitioners does not mention

[12] [1], p. 131.
[13] [1], p. 44.
[14] [1], p. 17.
[15] [1], p. 87.
[16] The SSACBK uses [7:4] as a reference for the Bell-LaPadula model (see [1], p. 41) and implies it is a reference for ORCON (see [1], p. 85). The seminal papers are not referenced.
[17] [1], p. 199.

teaching the principles and theoretical underpinnings of software assurance or security. Instead, it focuses on the students acquiring the skills needed to develop secure software. This is appropriate for training, but as noted above, is entirely inappropriate for graduate education, particularly in which students are expected to do research in the field of computer security.

IV. Approaches to Improving the SwACBK

Our goal in this section is to outline ways to make the SwACBK more suitable for guiding the development of curriculum at academic institutions. Our goal is *not* to make the SwACBK be a curriculum. A common body of knowledge, as noted above, is a collection of information upon which a curriculum could be built, and our suggestions are towards that end.

A. Context

An improved version of the SwACBK would spend some time discussing the dependencies of the software on the system, and in particular highlight the assumptions that the secure software makes about the system, its policies and procedures, and in general the environment in which it runs. It would then integrate this discussion into sections on software verification, validation, and evaluation.

This holistic view of security is central to the nature of security. As stated elsewhere, security is holistic; it depends on technical factors, but also upon social, organizational, procedural, and administrative factors. These often play a more critical role than technical factors, and should be treated with the same degree of importance as those factors. Specifically, these issues should be integrated throughout the SwACBK, for example by showing the role they play in secure software design and architectures.

One important step for the SwACBK would be to separate *functionality* from *assurance*. This would enable the reader to understand what portions of the SwACBK speak to what security services should be provided and how, and what parts of the SwACBK speak to demonstrating that the software meets its requirements, under the assumptions it makes.

B. Organization

The overall organization of the SwACBK is similar to that of the Software Engineering Common Body of Knowledge (SWEBOK). Both are organized around a model of the software development life cycle, with the SwACBK containing several sections particular to assurance. This organization is well suited to both bodies of knowledge.

Within individual chapters, the SwACBK can use principles to organize materials. Consider chapter 3, "Fundamental Concepts and Principles." This chapter's goal is to set forth the terms, concepts, and principles needed to develop, write, and deploy secure software. Such a chapter could be organized around "functionality" and "assurance." Falling into the first topic is confidentiality, integrity, and availability; falling into the second is dependability, accreditation, and certification. Functionality leads to a discussion of assets (what do you want to protect), risk (what are the risks if you do not protect your assets), and stakeholders (who decides what the assets and risks are). Assurance leads to discussions about what should be accredited, how does one gather evidence for assurance (and what evidence should be gathered), and what certifications are available and why are they important.

Secure software terminology can fit into the above discussion. For example, policies, cryptography, and architectural concepts fall under functionality; architectural concepts falls under assurance, because proper organization adds weight to assurance evidence. Similarly, software development processes may provide additional assurance evidence (either in favor of, or against, assurance claims).

Next come design principles; the SwACBK's selection of Saltzer and Schroeder is praiseworthy.

A brief history would round out this section nicely. The history should focus on the documents describing seminal work in the area, with a sentence or two describing the main ideas of each. Obvious choices for this section include the Ware report [16] and the Anderson report [8], among others (see section 4.3, "References," below.)

This organization would provide needed structure for the concepts, terms, and principles. Further, a curriculum developer could immediately focus on the main concepts of functionality and assurance, and structure the curriculum appropriately. The relevance of Saltzer's and Schroeder's principles would be apparent.

This point is central to our thesis: all classifications require some derivation, and cannot be laid out without justification. In some cases, a reference to an accepted organizational scheme is sufficient (such as one of the myriad of vulnerability or attack classifications), because the authors of that scheme have presented their rationale elsewhere. But the structure must be well-founded and sound, or the list becomes simply a collection of thoughts that may, or may not, be complete. Justifying such a

classification scheme is difficult at best, and at the worst overlooks relevant and important material.

Lists should be derived similarly. For example, consider §6.7, discussed above. The architectures in the itemized list should clearly indicate the relationships among the different styles. Reference monitors could be discussed first, and then the list broken into three lists. The first list covers supporting architectures (compartmentalization, layering, tolerant) built upon reference monitors, and should note these architectures overlap. The second list covers styles of architectures built upon elements of the first list (tolerant, adaptive, distributed). The third list describes the abstractions that can be built on the styles (MILS, MSLS, and so forth). Access control issues can follow, and the discussion can be framed in terms of the elements of the second and third lists supporting particular policies. A discussion of cross-domain control can use the problem of two different high-level policies to illustrate the need for specific mechanisms.

The key observation here is that the list of architectures must be more than a checklist—it must be organized to convey information about the relationships among the entities in the list. May organizations are possible. Whichever is chosen must be meaningful, and the reasons for its selection clear. In this way, the SwACBK can systematize the knowledge it contains.

C. Industry and Academic Examples

The authors of the SwACBK should draw on examples and problems from all spectra of society, not primarily government. In industry, motivation arises from sources such as the financial community, which must protect electronic representations of financial assets lest attackers steal money; from the legal community, which handles confidential information on a daily basis; and from the medical community, which deals with personal records that must not simply be kept secret, but also be protected from unauthorized change and be available in emergencies. In academia, financial aid offices must protect sensitive financial information, but must also make it available to potential funders; research labs need to protect data from alteration, falsification, and premature disclosure; and faculty and the administration must protect the integrity of grades recorded on computers. Any of these examples will motivate the need for software security in a wide variety of environments.

One overlooked aspect of security is personal security. Examples include protection of individual home computers from malware, or from trap doors in programs that transmit information illicitly. A classic case is a budgeting program that sends account information to an attacker. These examples would emphasize the personal nature of secure software.

D. References

The organization and selection of references is critical for the success of a common body of knowledge to be useful as a curriculum guide. The presence of references indicates either their importance (as a source document) or their clarity of exposition (as a teaching document). The SwACBK should indicate what each reference contributes, and why that reference is important.

An example of such a reference might be the Ware report, which first identified computer security as an important problem. In the history section of the SwACBK, an appropriate reference is:

> The Ware Report [Ware 1970] first discussed the importance of computer security as a policy, technical, and management problem.

The SwACBK has extensive lists of recommended references at the end of each chapter. The value of these references would be enhanced by a short sentence, similar to the above, for each reference. As an example, the Orange Book reference might be:

> [DoD 5200.28-STD 1985] DOD 5200.28-STD, *Department of Defense Trusted Computer System Evaluation Criteria*, 1985. First major computer security evaluation methodology, it influenced current evaluation technologies and criteria.

As an alternative, the references could be put in the sections for which they provide additional information. Either approach would place the reference in context, so the curriculum developer understands the benefits of using the reference.

E. Principles and Concepts

The SwACBK should include additional principles and concepts. The design principles of Saltzer and Schroeder are a good beginning, but these should be expanded to show how they underlie other concepts such as confinement, multi-level security, separation of duty, and the various methods of remediation such as recovery, tolerance, and interdiction.

Concepts such as reference validation mechanisms and reference monitors, requirements tracing and correspondence, and methods of justification are crucial to understanding how assurance works. The SwACBK should discuss these thoroughly. They are the foundation for security mechanisms, and developers who apply these

techniques to secure software development make a much stronger assurance case than those who do not.

V. Recommendations and Future Direction

The SwACBK is an ambitious undertaking. Its attempt to collect information into a common body of knowledge that defines what someone should know in order to write secure software is something that is badly needed.

The SwACBK is a first step, but—unfortunately—its goals are too lofty. Its authors want to provide a basis for an academic curriculum, for training, and for practitioners whose jobs range from developing secure software to acquiring and operating systems with that software. As constituted, the SwACBK contains information that each of these goals requires. But not all those goals require all the information; for example, the relevance of much of the information on acquisition is not something that a graduate curriculum would include, whereas training for purchasing agents might find that material essential. The material also lacks cohesiveness, primarily because the SwACBK treats the process of developing secure software as a sequence of independent steps. By considering each step in the process as being built upon a set of principles and previous steps, and showing how the process advances the development from requirements to deployment, validation in the environment, and operation, the SwACBK would provide a foundation for the common body of knowledge.

Acknowledgement. We gratefully acknowledge the support of NSF award CCR-0311723 to the University of California, Davis.

VI. References

[1] S. Redwine (ed), Secure Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software version 0.9 (draft), US Dept. of Homeland Security (Jan. 9, 2006).

[2] *Criteria for Accrediting Computing Programs Effective for Evaluations During the 2006–2007 Accreditation Cycle*, ABET Computing Accreditation Commission, Baltimore, MD 21202 (Feb. 9, 2006).

[3] *Committee on Courses of Instruction Policies and Procedures*, Academic Senate, University of California at Davis, Davis, CA 95616 (Jan. 2000); available at http://academicsenate.ucdavis.edu/ra/policy.htm.

[4] M. Bishop and D. Frincke, "Teaching Secure Programming," *IEEE Sec Priv* **3**(5) pp. 54–56 (2005).

[5] F. Cohen, "Information Systems Attacks: A Preliminary Classification Scheme," *Computers and Security* **16**(1) pp. 29–46 (1997).

[6] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proc IEEE* **63**(9) pp. 1278–1308 (1975).

[7] M. Bishop, *Computer Security: Art and Science*, Addison-Wesley Professional, Boston, MA (2003).

[8] J. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Electronic Systems Division, Hanscom Air Force Base, Hanscom, MA (1974).

[9] P. Szor, *The Art of Computer Virus Research and Defense*, Addison-Wesley Professional, Boston, MA (2005).

[10] R. Abbott, J. Chin, J. Donnelley, W. Konigsford, S. Tukubo, Shigeru, and D. Webb, "Security Analysis and Enhancements of Computer Operating Systems," NBSIR 76-1041, National Bureau of Standards, Washington DC (Apr. 1976).

[11] M. Vizedom, *Rites and Relationships: Rites of Passage and Contemporary Anthropology*, Sage Publications, Beverly Hills, CA (1976).

[12] E. Bersoff, V. Henderson, and S. Siegel., *Software Configuration Management*, Prentice-Hall, Englewood Falls, NJ (1980).

[13] T. Williams and K. Parker, "Design for Testability—A Survey," *IEEE Trans Comp* **C-31**(1) pp. 2–15 (1982).

[14] R. Patterson and D. Conner, "Building Commitment to Organizational Change," *Training and Development Journal* pp. 18–30 (Apr. 13, 1983).

[15] K. Thompson, "Reflections on Trusting Trust," *CACM* **27**(8) pp. 761–763 (1984).

[16] W. Ware, *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security*, Rand Report R609-1, RAND Corp., Santa Monica, CA 90407 (Feb. 1970).