Software Engineering Institute | Carnegie Mellon University

Foundations for Software Assurance

Carol Woody
Dan Shoemaker
Nancy Mead

July 2012

ABSTRACT: Our society's growing dependence on software makes the need for effective software assurance imperative. Motivation to address software assurance requires, at a minimum, an understanding of what to do, how to go about it, and why it is needed. Two key foundation elements are principles for software assurance and a curriculum to educate those who must address this need. This paper highlights efforts underway to address both of these elements.

1. DEFECTS ARE NOT AN OPTION IN TODAY'S WORLD

Computers are a vital part of our culture. In fact, it might be said that computers and the software that runs on them epitomize our modern society. Consider that 30 years ago you couldn't shop, bank, buy stocks online, play games or interact with people on a mobile device. Now all of that is possible, and the new opportunities that technology creates seem to multiply at an impossibly fast rate. Consequently, it is critically important to be able to trust the software that makes our way of life possible. Unfortunately, however, "commonly used software engineering practices permit dangerous defects that let attackers compromise millions of computers every year" [1]. Most of these defects are traceable to programming or design flaws, and they do not have to be actively exploited to be considered a threat [1]–[3]. These defects result from the fact that "commercial software engineering lacks the rigorous controls needed to [ensure defect free] products at acceptable cost" [1].

In fiscal terms, the exploitation of software defects costs the U.S. economy an average of $60 billion dollars annually [4]. Worse, it is estimated that "in the future, the Nation may [be at even greater risk] 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 significant concern is that the exploitation of a software flaw in a basic infrastructure component such as power or communication would lead to a significant national disaster [5]. The Critical Infrastructure Taskforce sums up such an event: "The nation's economy is increasingly dependent on cyberspace. This has introduced unknown interdependencies and single points of failure. A digital disaster strikes

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

some enterprise every day, [and] infrastructure disruptions have cascading impacts, multiplying their cyber and physical effects” [5].

Given the scope and potential impact of software defects, it is important to ensure the workforce follows proper software development and sustainment practices. The problem is that there is currently no authoritative point of reference to define what those practices are [3]. For that reason, in 2005 the U.S. Department of Homeland Security (DHS) created a group to define a common body of knowledge (CBK) for secure software assurance. The goal of the CBK was to itemize all the activities that might be involved in producing secure code. The aim was to then have the CBK serve as the basis for “defining workforce needs and competencies, leveraging sound practices, and guiding curriculum development for education and training relevant to software assurance” [3].

Nonetheless, the primary criticism of the CBK has been that, although it provides close to 300 pages of recommendations about what needs to be done, it provides very little specific information about why those practices are required and how they might apply in a range of situations. That is mainly because the goal of the CBK was to help the government ensure that it was getting secure software [6].

Subsequently, DHS enlisted the Carnegie Mellon Software Engineering Institute (SEI) to develop a curriculum for a Master of Software Assurance (MSwA) degree program [7]. The MSwA team drew on the CBK and other sources to develop a curriculum body of knowledge and associated outcomes.

In parallel with this effort, several MSwA curriculum authors pointed out the need for a seminal set of principles for secure software assurance [6], [8]–[9]. That need is what motivated a joint effort by a team at the SEI’s CERTÒ Program and the Software Engineering Program at Oxford University, UK. This research program was instituted in the fall of 2010 and has produced a coherent set of principles based on the efforts of those two groups. These principles are first reported here in section 2. As part of this effort, we created a mapping between the principles and the MSwA curriculum outcomes in order to validate our thinking (see section 5).

2. HISTORICAL PRINCIPLES FOR INFORMATION PROTECTION

Much of the information protection in place today is based on principles established by Saltzer and Schroeder in their paper titled “The Protection of Information in Computer Systems,” which appeared in Communications of the ACM in 1974 [10]. They defined security as “techniques that control who may use or modify the computer or the information contained in it" and described the three main categories of concern: confidentiality, integrity and availability (CIA). Their proposed design principles that focus on protection mechanisms to "guide the design and contribute to an implementation without security flaws" [10] are still taught in today's classrooms. They established eight principles for security in software design and development [10]:

"Economy of mechanism: Keep the design as simple and small as possible.

1. **Fail-safe defaults**: Base access decisions on permission rather than exclusion.
2. **Complete mediation**: Every access to every object must be checked for authority.
3. **Open design**: The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, and more easily protected, keys or passwords.
4. **Separation of privilege**: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.
5. **Least privilege**: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
6. **Least common mechanism**: Minimize the amount of mechanism common to more than one user and depended on by all users.
7. **Psychological acceptability**: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly."

Time has shown the value and utility in these principles; however, it is appropriate to consider that these were developed prior to the Morris worm that generated a massive denial of service by infecting over 6000 UNIX machines on November 2, 1988 [11]. To provide a technology context, consider that the IBM System 360 was in use from 1964–1978, and the IBM System 370 came on the market in 1972. An advanced operating system MVS (Multiple Virtual Storage) was released in March 1974 [12].

These principles were assembled prior to the identification of the more than 46500 software vulnerabilities and exposures that are currently exploitable in today's software products as described in the Common Vulnerabilities and Exposures (CVE) database at http://cve.mitre.org/. When these principles were developed, "buffer overflow," "malicious code," "cross-site scripting" and "zero-day vulnerabilities" were not part of the everyday vocabulary of operational software support personnel. Patches were carefully tested and scheduled to minimize operational disruption instead of pushed into operation to minimize attack vectors.

While these principles are still usable today in consideration of security within an individual piece of technology, they are no longer sufficient to address the complexity and sophistication of the environment within which that component must operate. We must broaden our horizon to consider the large scale, highly networked, software dependent systems upon which our entire critical infrastructure depends, from phones, power and water to industries such as banking, medicine and retail.

Software assurance is the commonly used term to describe this broader context. The Committee on National Security Systems (CNSS) [13] defines software assurance as follows:

> *"Software assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner."*

For purposes of developing the curriculum model for software assurance in the Master of Software Assurance Reference Curriculum report [7], the CNSS definition has been expanded as follows:

> *"Application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures."*

The expanded definition emphasizes the importance of both technologies and processes in software assurance, observes that computing capabilities may be acquired through services as well as new development, recognizes that security capabilities must be appropriate to the expected threat environment and identifies recovery from intrusions and failures as an important capability for organizational continuity and survival.

3. PRINCIPLES FOR SOFTWARE ASSURANCE

There are vast lists of practices and procedures that describe what should be done to address software assurance.[1] There are also an equal number of complaints that effective assurance is not being addressed in today's software. We posit that some of the inaction stems from a general lack of understanding about why this additional work is needed. In our scrutiny of the wide range of materials published, the case for why to focus on software assurance, a question any two-year-old would ask, has not yet been addressed. We propose the following seven principles in response:

1. **Risk**: A perception of risk drives assurance decisions. Organizations without effective software assurance perceive risks based on successful attacks to software and systems and usually respond reactively. They may implement assurance choices such as policies, practices, tools and restrictions based on their perception of the threat of a similar attack and the expected impact should that threat be realized. Organizations can incorrectly perceive risk when they do not understand their threats and impacts. Effective software assurance requires that risk knowledge be shared among all stakeholders and technology participants; however, too frequently, risk information is considered highly sensitive and is not shared, resulting in uninformed organizations making poor risk choices.
2. **Interactions**: Highly connected systems like the Internet require alignment of risk across all stakeholders and all interconnected technology elements; otherwise, critical threats will be missed or ignored at different points in the interactions. It is no longer sufficient only to consider highly critical components when everything is highly interconnected. Interactions occur at many technology levels (e.g., network, security appliances, architecture, applications, data storage, etc.) and are supported by a wide range of roles. Protections can be applied at each of these points and may conflict if not well orchestrated. Because of interactions, effective assurance requires that all levels and roles consistently recognize and respond to risk.
3. **Trusted Dependencies**: Because of the wide use of supply chains for software, assurance of an integrated product depends on other people's assurance decisions and the level of trust placed on these dependencies. The integrated software inherits all of the assurance limitations of each interacting component. In addition, unless specific restrictions and controls are in place, every operational component including infrastructure, security software and other applications can be affected by the assurance of every other

[1] For a starting point see https://buildsecurityin.us-cert.gov/swa/.

component. There is a risk each time an organization must depend on others' assurance decisions. Organizations should decide how much trust they place in dependencies based on a realistic assessment of the threats, impacts and opportunities represented by an interaction. Dependencies are not static, and trust relationships should be regularly reviewed to identify changes that warrant reconsideration. The following examples describe assurance losses resulting from dependencies:

- Defects in standardized pieces of infrastructure (such as operating systems, development platforms, firewalls, routers, etc.) can serve as widely available threat entry points for applications.
- Using many standardized software tools to build technology establishes a dependency for the assurance of the resulting software product. Vulnerabilities can be introduced into software products by the tool builders.

4. **Attacker**: A broad community of attackers with growing technology capabilities are able to compromise the confidentiality, integrity and availability of an organization's technology assets. There are no perfect protections against attacks, and the attacker profile is constantly changing. The attacker will use technology, processes, standards and practices to craft a compromise (known as a socio-technical responses). Attacks are crafted to take advantage of the ways we normally use technology or designed to contrive exceptional situations where defenses are circumvented.
5. **Coordination and Education**: Assurance requires effective coordination among all technology participants.. Protection must be applied broadly across the people, processes and technology in an organization because the attacker will take advantage of all possible entry points. Authority and responsibility for assurance must be clearly established at an appropriate level in the organization to ensure the organization effectively participates in software assurance. This assumes that all participants know about assurance, and that is not usually a reality. There is much to be done to educate people on software assurance.
6. **Well Planned and Dynamic**: Assurance must represent a balance among governance, construction and operation of software and systems and is highly sensitive to changes in each of these areas. An adaptive response is required for assurance because the applications, interconnections, operational usage and threats are always changing. Assurance is not a once-and-done activity. It must continue beyond the initial operational implementation through operational sustainment. Assurance cannot be added later; it must be built to the level of acceptable assurance that organizations need. No one has resources to redesign systems every time the threats change, and assurance cannot be readily adjusted upward after the fact.

7. **Measurable**: A means to measure and audit overall assurance must be built in. That which is not measured cannot be managed. Each stakeholder or technology user will address only the assurance for which they are held accountable. Assurance will not compete successfully with other competing needs unless results are monitored and measured. All elements of the socio-technical environment, including practices, processes and procedures, must be tied together to evaluate operational assurance. Organizations with more successful assurance measures react and recover faster, learn from their reactive responses and that of others and are more vigilant in anticipating and detecting attacks. Defects per lines of code, a common development measure, may be useful for code quality but are not sufficient evidence for overall assurance because they provide no perspective on how that code behaves in an operational context. Both focused and systemic measures are needed to ensure the components are engineered with sound security and the interaction among components establishes effective assurance.

Risk management has been widely studied. There are several organizational and cultural challenges that contribute to how an organization addresses risk. Because of the importance of risk in software assurance, these challenges will contribute to a successful assurance outcome [14]:

- Open communication: Risks cannot be addressed if they are not communicated to and understood by the decision makers. Evaluation activities must be built upon collaborative approaches that encourage the exchange of security and risk information among all levels of the organization.
- Culture of sharing: When participants have a culture of sharing, there is a greater likelihood that information important to assurance will be effectively communicated; when this sharing includes formal documentation, there is a greater likelihood that the information will persist.
- Traditional boundaries are potential barriers to communication but not to risk: Organizational, system, contract and classification boundaries may inhibit critical communication of risks, threats, impacts, measures, etc. critical to software assurance.
- Complexity increases the challenges for assurance and must be managed through the application of effective software engineering.

4. MASTER OF SOFTWARE ASSURANCE CURRICULUM

To address the disconnect between research, education and practical development of assured software, the DHS National Cyber Security Division (NCSD) enlisted the SEI to develop a curriculum for a Master of Software Assurance degree program and define transition strategies for future implementation. As noted in the curriculum report, the need for a master's level program in this discipline has been growing for years [7]:

- "A study by the nonpartisan Partnership for Public Service points out that, 'The pipeline of new talent [with the skills to ensure the security of software systems] is inadequate. . . . only 40 percent of CIOs [chief information officers], CISOs [chief information security officers] and IT [information technology] hiring managers are satisfied or very satisfied with the quality of applicants applying for federal cybersecurity jobs, and only 30 percent are satisfied or very satisfied with the number of qualified candidates who are applying'" [15].
- "The need for cybersecurity education was emphasized in the New York Times when Dr. Nasir Memon, a professor at the Polytechnic Institute of New York University, was quoted as saying, 'There is a huge demand, and a lot more schools have created programs, but to be honest, we're still not producing enough students'" [16].
- "In discussions with industry and government representatives, we have found that the need for more capacity in cybersecurity continues to grow. Anecdotal feedback from the MSwA curriculum development team members' own students indicates that even a single course with a cybersecurity focus enhances their positioning in the job market. They felt that they were given job offers they would not have received otherwise" [7].

Another aspect of the need for cybersecurity education occurs in educational institutions. The curriculum authors point out, based on their collective experience in software engineering education, that it can be very difficult to start a new program or track from scratch. The authors offer assistance to those organizations and faculty members who wish to undertake such an endeavor. The objective is to support their needs, while recognizing that there are a variety of implementation strategies. Each participant must select what works best within their institution and for their students.

While information security is important, academic programs in information security typically focus on system administrator activities for operational systems, whereas the focus in the MSwA curriculum was on systems under development. Software engineering provides much excellent foundational material, and all the curriculum development team members have a software engineering background. However, the authors recognized that development of assured software needs to go beyond good software engineering practice, and, indeed, the resulting curriculum reflects this.

4.1 MASTER OF SOFTWARE ASSURANCE EDUCATION OUTCOMES

The Master of Software Assurance (MSwA) Reference Curriculum report recommends a core body of knowledge (CBK) that includes seven outcome areas. Brief descriptions for each outcome taken from the report follow. The MSwA CBK and more detailed descriptions of the outcomes can be found in the curriculum report [7].

Outcome 1. Assurance Across Life Cycles: Graduates will have the ability to incorporate assurance technologies and methods into life-cycle processes and development models for new or evolutionary system development, and for system or service acquisition.

Outcome 2. Risk Management: Graduates will have the ability to perform risk analysis, tradeoff assessment, and prioritization of security measures.

Outcome 3. Assurance Assessment: Graduates will have the ability to analyze and validate the effectiveness of assurance operations and create auditable evidence of security measures.

Outcome 4. Assurance Management: Graduates will have the ability to make a business case for software assurance, lead assurance efforts, understand standards, comply with regulations, plan for business continuity, and keep current in security technologies.

Outcome 5. System Security Assurance: Graduates will have the ability to incorporate effective security technologies and methods into new and existing systems.

Outcome 6. System Functionality Assurance: Graduates will have the ability to verify new and existing software system functionality for conformance to requirements and absence of malicious content.

Outcome 7. System Operational Assurance: Graduates will have the ability to monitor and assess system operational security and respond to new threats.

5. MAPPING MSWA CURRICULUM OUTCOMES TO PRINCIPLES

For education in software assurance to be effective, it must support the principles that have been identified as critical to effective software assurance. This section describes how the curriculum outcomes effectively map to the principles for software assurance described in section 2. It also validates the completeness of the principles. It is useful to note that the principles and curriculum model were developed independently and the mapping was done afterward.

Principle 1. Risk
- Outcome 2. Risk Management

Principle 2. Interactions
- Outcome 2. Risk Management

Principle 3. Trusted Dependencies
- Outcome 2. Risk Management

Principle 4. Attacker
- Outcome 5. System Security Assurance
- Outcome 6. System Functionality
- Outcome 7. System Operational Assurance

Principle 5. Everyone is involved
- Outcome 4. Assurance Management
- Outcome 5. System Security Assurance

Principle 6. Assurance must be dynamic
- Outcome 1. Assurance Across Life Cycles
- Outcome 5. System Security Assurance
- Outcome 6. System Functionality Assurance
- Outcome 7. System Operational Assurance

Principle 7. Assurance must be measurable
- Outcome 3. Assurance Assessment

6. CONCLUSIONS AND FUTURE PLANS

The principles will help everyone involved in software assurance understand its importance and value. Communicating them across the software development community is a critical next step. Identifying memory aids similar to “CIA” will make this task easier and support longer retention.

Both the Association of Computing Machinery and the IEEE Computer Society have recognized the MSwA curriculum as appropriate for a master’s program in software assurance. Efforts are underway to develop courseware for faculty to enhance adoption of the curriculum. In addition, the team has assembled research and sample course outlines that can be used at the undergraduate and community college levels to introduce topics related to software assurance earlier in the educational process. The undergraduate material was published [17] with the MSwA curriculum [7], and the community college material was published in fall 2011 [18]. Software engineering practitioners should urge their alma maters to consider incorporating this program; faculty interested in developing this master's program should contact us for further information and assistance.

REFERENCES

[1] President's Information Technology Advisory Committee, "Cybersecurity: A Crisis of Prioritization," Executive Office of the President, National Coordination Office for Information Technology Research and Development, Arlington, 2005.

[2] C. Jones, "Software Quality in 2005: A Survey of the State of the Art," Software Productivity Research, Marlborough, 2005.

[3] S. T. Redwine, Ed., "Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire and Sustain Secure Software, Version 1.1," U.S. Department of Homeland Security, Washington, 2006.

[4] M. Newman, "Software Errors Cost U.S. Economy $59.5 Billion Annually," National Institute of Standards and Technology (NIST), Gaithersburg, 2002.

[5] R. A. Clark and H. A. Schmidt, "A National Strategy to Secure Cyberspace,"The President's Critical Infrastructure Protection Board, Washington, 2002.

[6] M. Bishop and S. Engle, "The Software Assurance CBK and University Curricula," in *Proceedings of the 10th Colloquium for Information Systems Security Education*, University of Maryland, University College, Adelphi, MD June 2006.

[7] N. R. Mead, et al., "Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-2010-TR-005, 2010.

[8] S. T. Redwine, "Toward an Organization for Software System Security Principles and Guidelines," James Madison University, 2008.

[9] K. M. Goertzel, "Introduction to Software Security," *Build Security In*, Department of Homeland Security.

[10] J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” *Communications of the ACM*, vol. 17, issue 7, 1974.

[11] Wikipedia, “Morris worm,” *Wikipedia*. Available: http://en.wikipedia.org/wiki/Morris_worm.

[12] Wikipedia, “IBM System/370,” Wikipedia. Available: http://en.wikipedia.org/wiki/System/370.

[13] Committee on National Security Systems, “Instruction No. 4009,” *National Information Assurance Glossary.* Revised June 2009.

[14] A. Dorofee, J. Walker, C. Albert, R. Higuera, R. Murphy, and R. Williams, *Continuous Risk Management Guidebook*, Pittsburgh: Carnegie Mellon University, 1996, pp. 7-9

[15] A. Dorofee, J. Walker, C. Albert, R. Higuera, R. Murphy, and R. Williams, *Continuous Risk Management Guidebook*, Pittsburgh: Carnegie Mellon University, 1996, pp. 7-9. Available: http://ourpublicservice.org/OPS/publications/viewcontentdetails.php?id=135

[16] C. Drew, “Wanted: ‘Cyber Ninjas.’” *New York Times*, 2009*.*

[17] N. R. Mead, et al., “Software Assurance Curriculum Project Volume II: Undergraduate Course Outlines,” Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-2010-TR-019.

[18] N. R. Mead, et al., “Software Assurance Curriculum Project Volume IV: Community College Education,” Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-2011-TR-017.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Department of Homeland Security or the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM-0001120