Software Engineering Institute

# Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum

Nancy R. Mead, Software Engineering Institute
Julia H. Allen, Software Engineering Institute
Mark Ardis, Stevens Institute of Technology
Thomas B. Hilburn, Embry-Riddle Aeronautical University
Andrew J. Kornecki, Embry-Riddle Aeronautical University
Richard Linger, Software Engineering Institute
James McDonald, Monmouth University

**August 2010**

**TECHNICAL REPORT**
CMU/SEI-2010-TR-005
ESC-TR-2010-005

**CERT® PROGRAM**


http://www.cert.org

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.



This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

# Table of Contents

# List of Figures

# Acknowledgments

The authors thank the following individuals for their contributions to this report. We greatly appreciate their insights and efforts.

- Our sponsor Joe Jarzombek, U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD), had the insight to recognize the need for such a curriculum and support its development.
- These individuals provided valuable feedback through their responses to our questionnaire on software assurance education.
  - Warren Axelrod, Delta Risk
  - Jennifer L. Bayuk, Information Security Specialist
  - John Carlson, Morgan Stanley
  - Mary Ann Davidson, Oracle
  - Cassio Goldschmidt, Symantec
  - Eric Guerrino, BNY Mellon
  - Frank Gutcher, The Boeing Company
  - Jim Krodel, Pratt & Whitney Aircraft
  - Andrew McGettrick, University of Strathclyde, Scotland, UK
  - Paul N. Smocer, BITS & FSTC, Financial Services Roundtable
- These individuals worked with us on the planning and execution of the work in the early stages.
  - Michael Ryan, Dublin City University
  - Dan Shoemaker, University of Detroit Mercy
- These individuals provided critical insights in their review of this document.
  - Warren Axelrod, Delta Risk
  - Carol Sledge, Software Engineering Institute
  - Yair Levy, Nova Southeastern University

We also acknowledge the work done by the Integrated Software & Systems Engineering Curriculum (iSSEc) Project on the *Graduate Software Engineering 2009 (GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering, Version 1.0* document [iSSEc 2009]. We also acknowledge the Department of Homeland Security's work on the Software Assurance Curriculum Body of Knowledge (SwACBK) [DHS 2010B]. The GSwE2009 and SwACBK are part of the foundation upon on which we based the Master of Software Assurance curriculum. The DHS NCSD Workforce Education & Training Working Group provided valuable review comments on the draft curriculum document.

In addition, we thank the following individuals from the Software Engineering Institute for their support: David Biber, Jennifer Kent, and Tracey Tamules.

# Executive Summary

Modern society is deeply and irreversibly dependent on software systems of remarkable scope and complexity in areas including defense, government, energy, communication, transportation, manufacturing, and finance. The security and correct functionality of these systems are absolutely vital; poor or absent security and incorrect functionality can have devastating consequences including loss of life. Yet these software systems (and systems of systems) continue to exhibit errors and vulnerabilities and are regularly subject to attack and compromise. Attacker actions can result in severe impacts and losses for the organizations that build, deploy, and operate these systems, as well as the business partners and customers that use them.

Recognizing these realities, the U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD) enlisted the resources of the Software Engineering Institute at Carnegie Mellon University to develop a reference curriculum for a Master of Software Assurance degree program and define transition strategies for future implementation. This report is Volume I of the project. Volume II focuses on an undergraduate curriculum specialization for software assurance [Mead 2010].

For the purposes of this curriculum, the discipline of software assurance is targeted specifically to the security and correct functionality of software systems, whatever their origins, subject matter, or operational environments. The need for a master's level program in this discipline has been growing for years.

The purpose of this Master of Software Assurance Curriculum project is to identify and present a core body of knowledge from which to create such a degree program, as a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. The foundation upon which this work rests includes the *Graduate Software Engineering 2009 (GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering* [iSSEc 2009], work on the DHS Security Build Security In website by Carnegie Mellon University's Software Engineering Institute [DHS 2010a], the Software Assurance Curriculum Body of Knowledge (SwACBK) [DHS 2010b], and the authors' discussions and professional experience. Authors of this curriculum include faculty and researchers from Carnegie Mellon University, Embry-Riddle Aeronautical University, Monmouth University, and Stevens Institute of Technology.

The primary audience for the Master of Software Assurance Curriculum (MSwA2010) is faculty who are responsible for designing, developing, and maintaining graduate software engineering programs that have a focus on software assurance knowledge and practices.

For purposes of the MSwA2010 curriculum defined in this report, the definition of software assurance has been extended from the generally accepted one offered by the Committee on National Security Systems [CNSS 2009]. This is the expanded definition:

> *Application of technologies and processes to achieve a required level of confidence[1] 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 extended 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 and evolution, 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. This definition is expanded in Section 2 by decomposing it into its constituent components and concepts.

Areas of special emphasis and unique properties (shown in italics) that distinguish this curriculum from traditional software engineering and computer science programs include a focus on

- software *and services*
- development *and acquisition*
- *security* and correct functionality
- *software analytics*
- *system operations*
- *auditable evidence*

The authors performed seven steps to develop the MSwA2010 curriculum content, including developing project guidelines; identifying credible, reputable sources to consider; selecting life-cycle phase topics (such as requirements engineering) and organizing candidate practices and categories in these topic areas; soliciting external feedback from recognized faculty, thought leaders, and practitioners; and developing curriculum outcomes and the core body of knowledge.

Because of the technical nature of software assurance, we anticipate that students entering an MSwA program will possess undergraduate degrees in disciplines such as computer science; software engineering; electrical, electronic, and computer engineering; mathematics; or information systems. We present a list of required and desired prerequisites in three categories: computing foundations, software engineering, and security engineering. These prerequisites will likely be satisfied through some combination of undergraduate courses, work experience, and possibly remedial education prior to the start of an MSwA program.

The outcomes (knowledge, skills, and capabilities) that faculty members can use to structure and guide curriculum development and that graduates can expect after completing the program are organized into the following topics:

- Assurance Process and Management
  - Assurance Across Life Cycles
  - Risk Management
  - Assurance Assessment
  - Assurance Management

[1] In the CNSS definition, the use of the word “confidence” implies that there is a basis for the belief that software systems and services function in the intended manner.

- Assurance Product and Technology
  - System Security Assurance
  - System Functionality Assurance
  - System Operational Assurance

In developing the core body of knowledge (BoK), each outcome was captured as a knowledge area, and each knowledge area was subdivided into a set of knowledge units with assigned cognitive levels from an education classification system (the Bloom's Taxonomy system, as described in Appendix A).

Using the MSwA2010 BoK, the curriculum architecture identifies the minimum content that all degree programs should include. The architecture and course structures can be used to organize and package the body of knowledge. The MSwA2010 BoK provides for preparatory content, core course content, elective content, and a capstone experience through which students can demonstrate their understanding and ability to apply what they have learned. The curriculum architecture is similar to the one proposed in the GSwE2009 and is compatible with software engineering master's programs that are based on the GWsE2009 curriculum. The MSwA2010 curriculum is intended to provide a structural basis for programs that deliver the outcomes described.

Having a defined set of student prerequisites, established outcomes, a core body of knowledge, and a curriculum architecture is necessary but not sufficient. Often the most challenging part of putting a new program or a new track in place is implementation. This report provides several guidelines and recommendations for faculty members to consider when contemplating such a program. These recommendations include suggestions for planning and launching a new program, recruiting and preparing students, finding and training faculty, acquiring resources, and teaching capstone courses effectively.

This report closes with a description of the additional activities that are needed to support disseminating information about the MSwA2010 curriculum and transitioning it into both new and existing degree programs and tracks. In order for this work to be considered successful, the curriculum must be available, understood by the targeted academic and hiring communities, viewed as a key reference for software assurance curriculum development, and used in the development and modification of software-assurance-focused curricula.

# Abstract

Modern society depends on software systems of ever-increasing scope and complexity in virtually every sphere of human activity, including business, finance, energy, transportation, education, communication, government, and defense. Because the consequences of failure can be severe, dependable functionality and security are essential. As a result, software assurance is emerging as an important discipline for the development, acquisition, and operation of software systems and services that provide requisite levels of dependability and security.

This report is the first volume in the Software Assurance Curriculum Project sponsored by the U.S. Department of Homeland Security. This report presents a body of knowledge from which to create a Master of Software Assurance degree program, as both a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. The report details the process used to create the curriculum and presents the body of knowledge, curriculum architecture, student prerequisites, and expected student outcomes. It also outlines an implementation plan for faculty and other professionals who are responsible for designing, developing, and maintaining graduate software engineering programs that have a focus on software assurance knowledge and practices. The second volume, *Undergraduate Course Outlines* (CMU/SEI-2010-TR-019), presents seven course outlines that could be used in an undergraduate curriculum specialization for software assurance.

# 1 The Software Assurance Curriculum Project

The purpose of the Master of Software Assurance Curriculum project (MSwA2010) is to develop and present a core body of knowledge (BoK) from which to create a master's level degree program in software assurance, as a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. The foundation upon which this work rests includes the *Graduate Software Engineering 2009 (GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering* [iSSEc 2009] and work on the U.S. Department of Homeland Security Build Security In website done by Carnegie Mellon University's (CMU) Software Engineering Institute (SEI) [DHS 2010a]. Authors of this curriculum include faculty and researchers from CMU, Embry-Riddle Aeronautical University, Monmouth University, and Stevens Institute.

## The Need for Software Assurance Education

Modern society is deeply and irreversibly dependent on software systems of remarkable scope and complexity in areas including defense, government, energy, communication, transportation, manufacturing, and finance. The security and correct functionality of these systems are absolutely vital. Yet they continually exhibit errors and vulnerabilities, and they are regularly subject to attack and compromise with potentially severe consequences for the organizations that build, deploy, and operate them, as well as the business partners and customers that use them.

Recognizing these realities, the U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD) enlisted the resources of the CERT®[2] Program at the SEI to develop a curriculum for a Master of Software Assurance degree program and define transition strategies for implementation. For the purposes of this curriculum, the discipline of software assurance is targeted specifically to the security and correct functionality of software systems, whatever their origins, subject matter, or operational environments. The need for a master's level program in this discipline has been growing for years.

- At the Knowledge Transfer Network Workshop in Paris in March 2009, cybersecurity education was recognized as part of the information security, privacy, and assurance roadmap vision and as one of its lines of development [LSEC 2009].
- A study by the nonpartisan Partnership for Public Service points out that "[President Obama's] success in combating these threats [to cybersecurity] and the safety of the nation will depend on implementing a comprehensive and coordinated strategy—a goal that must include building a vibrant, highly trained and dedicated cybersecurity workforce in this country." The report found 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" [PPS 2009].

[2] ® CERT is a registered mark owned by Carnegie Mellon University.

- The New York Times emphasized the need for cybersecurity education in quoting Dr. Nasir Memon, a professor at the Polytechnic Institute of New York University: "There is a huge demand, and a lot more schools have created programs, but to be honest, we're still not producing enough students" [Drew 2009].
- CMU and CERT have been active in the software assurance area for years, particularly in the Survivability and Information Assurance (SIA) Curriculum and the Scholarship for Service program [CERT 2007]. The SIA Curriculum has been provided to thousands of faculty members and other interested parties. The Federal Cyber Service's Scholarship for Service program offers scholarships to applicants who attend an approved institution of higher learning and agree to work for several years in the cybersecurity area at U.S. government organizations after graduation [OPM 2010]. The popularity and growth of this program is an indicator of the pressing need for cybersecurity expertise.
- 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 authors' own students indicates that even a single course with a cybersecurity focus enhances the students' positioning in the job market. Students felt they were made job offers they would not have received otherwise.
- Another aspect of the need occurs in educational institutions that need assistance in starting a cybersecurity concentration. Based on our collective experience in software engineering education, we know that it can be very difficult to start a new program or track from scratch, so we plan to assist those organizations and faculty members that wish to undertake such an endeavor. Our objective is to support their needs while recognizing that there are a variety of implementation strategies.

### Definition of Software Assurance

In developing a curriculum for software assurance, it is important to start with a clear and concise definition of the discipline. The Committee on National Security Systems defines software assurance as follows [CNSS 2009]:

> *Software assurance (SwA) is the level of confidence[3] 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 the curriculum defined in this report, the CNSS definition has been expanded as follows:

> *Application of technologies and processes to achieve a required level of confidence[3] 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

[3] In the CNSS definition, the use of the word "confidence" implies that there is a basis for the belief that software systems and services function in the intended manner.

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.

### Audience

The primary audience for the MSwA2010 curriculum is faculty who are responsible for designing, developing, and maintaining graduate software engineering programs that have a focus on software assurance knowledge and practices. In addition, the MSwA2010 project will likely interest those in development and acquisition organizations who have responsibility for staffing positions in software assurance and for providing their software engineers with increased software assurance capabilities. The MSwA2010 project also provides a model to those who assess software-assurance-oriented programs for curriculum organization, content, outcomes, and support.

### Scope

The scope of this report is to identify a curriculum BoK for a master's degree in software assurance, or for a software assurance track within a master's degree program in computer science or software engineering. We developed this material intending that this would be a degree for practitioners, not for researchers, and we did not consider, for example, the content of a doctoral program in software assurance. Because it is likely that there may be overlap between upper-division undergraduate courses and first-year graduate courses in software assurance, we considered undergraduate coursework in software assurance in a separate report, *Software Assurance Curriculum Project Volume II: Undergraduate Course Outlines* [Mead 2010]. It is also possible that this material could be used to develop continuing education or certificate programs in both government and industry. Because of our affiliations, this report is U.S.-centric, although we would welcome adaptation of this material for use internationally or enhancement of the report to include international programs.

Areas related to software assurance, such as software safety, reliability, and dependability, as well as software process and management models, were not the primary focus of this project. We recognize that these areas provide important contributions to software assurance; the curriculum builds on and in some areas extends these capabilities. Information assurance (distinct from software assurance) is also not the primary focus of this report. Although we consider protection of information in deployed software to be important, we believe that this has been adequately addressed by existing education and training programs. To the extent that data is part of a software system, we are concerned with data insofar as it is related to software assurance.

### Comparison to Other Programs

The curriculum described in this report can be offered as an independent master's degree program in software assurance. It can also be offered as a track in a Master of Software Engineering (MSE) or a Master of Computer Science degree program. This report describes how it can be incorporated as a track in an MSE degree program if the software engineering program is based on the GSwE2009 recommendations. We envision that it could be incorporated as a track in other degree programs as well, but we have not yet done the needed analysis to support it. The independent master's degree program in software assurance we describe assumes a student enters

the program with an undergraduate degree in computer science [ACM 2008], computer engineering [IEEE-CS 2004a], or software engineering [IEEE-CS 2004b] and supplements the content of those degrees with appropriate prerequisite materials. For students with other backgrounds, the program incorporates the necessary portions of computer science and software engineering preparatory material to allow them to study software assurance.

### Organization of This Report

Section 2 expands the definition of software assurance by decomposing it into its constituent components and concepts. This section describes unique properties of the MSwA2010 curriculum that distinguish it from traditional software engineering and computer science programs. It then presents a description of the seven-step process that we used to develop MSwA2010 curriculum content, including Figures 1 and 2, which present high-level and detailed views of the MSwA2010 development process. Readers who are interested in the results of this report, but not the process used to arrive at it, can skip this section.

Section 3 presents MSwA2010 project guidelines that were used to establish the foundation, scope, and boundaries for project activities and decision making. These 14 guidelines, both strategic and tactical, draw heavily from the GSwE2009. Readers who are interested in the results of this report, but not the guidelines used by the curriculum development team, can skip this section.

Section 4 defines the outcomes (knowledge, skills, and capabilities) that faculty members can use to structure and guide curriculum development and that graduates can expect after completing the program. Outcomes are organized into the following knowledge areas:

- Assurance Process and Management
  - Assurance Across Life Cycles
  - Risk Management
  - Assurance Assessment
  - Assurance Management
- Assurance Product and Technology
  - System Security Assurance
  - System Functionality Assurance
  - System Operational Assurance

Section 5 describes required and desired prerequisite knowledge and skills that students of an MSwA program should have mastered prior to starting an MSwA program. Because of the technical nature of software assurance, it is anticipated that entrants to an MSwA program will possess undergraduate degrees in disciplines such as computer science; software engineering; electrical, electronic, and computer engineering; mathematics; or information systems. Prerequisites will likely be satisfied through some combination of undergraduate courses, work experience, and possibly remedial education prior to the start of an MSwA program.

Section 6 presents candidate architectures and course packaging that can be used to organize the MSwA2010 BoK to achieve the outcomes described in Section 4. The architecture provides for preparatory content, core course content, elective content, and a capstone experience through

which students can demonstrate their understanding and ability to apply what they have learned. The curriculum architecture is similar to the one proposed in the GSwE2009 and is compatible with software engineering master's programs that are based on that curriculum.

Section 7 describes the core BoK for the MSwA2010 curriculum. It is structured into seven knowledge areas (as listed in Section 4), with each knowledge area subdivided into a set of knowledge units. The knowledge units are defined in terms of the Bloom cognitive levels, an educational classification system (refer to Appendix A).

Section 8 includes a number of issues that faculty members need to address when implementing any new academic program, including a master's program in software assurance. The issues include planning and launching a new program, recruiting and preparing students, finding and training faculty, acquiring resources, and teaching capstone courses effectively.

Section 9 closes the report by describing the additional activities that are needed to support disseminating and transitioning the MSwA2010 curriculum into degree programs and tracks, both new and existing. In order for this work to be considered successful, the curriculum must be available, understood by the targeted academic and hiring communities, viewed as a key reference for software assurance curriculum development, and used in the development and modification of software-assurance-focused curricula.

Details on Bloom's Taxonomy and its application to the MSwA2010 BoK, software development life-cycle (SDLC) practices and their relationship to the core BoK, a summary of responses to our external questionnaire, and other supporting details are contained in the appendices. Note that Appendix B contains an extensive bibliography, which will be of interest to educators who implement an MSwA degree program.

# 2 Curriculum Project Foundations

To lay the foundation for the MSwA2010 curriculum project, we expanded the definition of software assurance by decomposing it into its constituent components and concepts. This expansion and clarification sets the boundary for what is considered in and out of scope for the curriculum. We highlight unique properties of the MSwA2010 curriculum that distinguish it from traditional software engineering and computer science programs while also pointing out commonalities. We close this section with a description of the seven-step process that was used to develop MSwA2010 curriculum content.

### Implications of the Definition of Software Assurance for Curriculum Development

The definition of software assurance we use in this report is as follows:

> *Application of technologies and processes to achieve a required level of confidence*[4] *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.*

This definition provides overarching guidance for the MSwA2010 curriculum development. It is thus illuminating to parse the definition into its constituent components as a first step in understanding the objectives and ultimate structure of the curriculum.

- focus on both *software systems and services*

  Software capabilities can originate from many sources, including new system development; legacy system evolution; system acquisition through a variety of means, including supply chains, open source, and commercial, off-the-shelf (COTS); and service acquisition through methods including service-oriented architecture (SOA), cloud computing, and virtualization. Systems often aggregate combinations of these sources, all of which require a level of assurance with respect to correct functionality and security. In some cases, such as service acquisition, the software itself may not be available for analysis, and assurance must be achieved through other means. *Thus, the MSwA2010 curriculum must focus on both software systems and services in meeting assurance objectives.*

- software systems and services *function in the intended manner*

  Software systems and services must exhibit levels of quality and correct functionality commensurate with the consequences of their failure. Developing quality software requires rigorous software engineering capabilities and best practices in technologies and processes. Effective development, testing, and management skills are always required. Software assurance adds key perspectives and capabilities to development and acquisition processes to further improve quality. *Thus, the MSwA2010 curriculum must include technologies and processes to achieve correct functionality and reduce errors in software development and evolution, as well as in software and service acquisition.*

[4] In the CNSS definition, the use of the word “confidence” implies that there is a basis for the belief that software systems and services function in the intended manner.

- software systems and services *are free from accidental or intentional vulnerabilities*

  In operational use, both legitimate users and intruders seeking to disrupt operations or obtain access to information use software systems. Intruders seek vulnerabilities in software they can use to gain access and control. Avoiding vulnerabilities (where possible) and eliminating vulnerabilities (where necessary) require thoroughly analyzing software and applying rigorous security requirements engineering, architecture and design, coding, and testing techniques. *Thus, the MSwA2010 curriculum must focus on the development of robust software systems and the acquisition of software services that do not provide means to achieve unauthorized access and exploitation of vulnerabilities.*

- software systems and services *provide security capabilities appropriate to the threat environment*

  Software systems operate in threat environments whose virulence can vary with the value of the functions and information the systems provide. High-value systems will be subjected to sophisticated attacks at all levels and must incorporate security capabilities to ensure that intrusion is as difficult and costly as possible to the intruder. Virtually all systems must implement security capabilities, such as authentication, authorization, non-repudiation, and privacy, and support the properties of availability, confidentiality, and integrity. *Thus, the MSwA2010 curriculum must include threat environment analysis and security assurance technologies and methods at application, system, and network levels. The curriculum must also include methods for assuring security in the acquisition of software and services and for monitoring security in system operations.*

- software systems and services *recover from intrusions and failures*[5]

  No amount of security and discipline can guarantee that systems will not be exploited and compromised. Operational continuity and survival must be assured even in adverse circumstances. *Thus, the MSwA2010 curriculum must include methods to define and assure that capabilities exist to recover from intrusions, failures, and accidents.*

These objectives are to be achieved through the following means:

- *application of technologies and processes*

  Assurance technologies include analytical areas such as verifying software functionality; analyzing software vulnerabilities, threat environments, and security capabilities; and reverse engineering software to determine as-built functionality and security properties. Assurance processes define methods for achieving required levels of confidence that can be integrated into traditional software development and acquisition process models. *Thus, in addition to a technology focus, the MSwA2010 curriculum must include a process-oriented view of assurance activities, including organizational goals, objectives, and constraints; risk analysis and reduction; and integration of assurance processes into organizational processes, methods, and procedures.*

[5] Includes recovery from accidents as well.

- achieve a *required level of confidence* that assurance goals are met

  A key responsibility of software assurance is to create auditable evidence that supports achievement of assurance goals. Assurance requirements can vary with business objectives, threat environments, system capabilities, risk analysis, legal and compliance requirements, and internal and external standards. *Thus, the MSwA2010 curriculum must provide methods for cost-effective and auditable assurance that satisfy organizational and technical objectives, requirements, and constraints.*

### Principal Focus Areas for MSwA2010 Curriculum Development

This analysis of the definition of software assurance highlights areas of special emphasis for the MSwA2010 curriculum. This analysis also reveals differences and commonalities with traditional computer science and software engineering curricula.

- focus on software *and services*

  Many organizations obtain computing capabilities through contracted services. The MSwA2010 curriculum must address correct functionality and security of services when the software itself may be unavailable. Software services are typically not emphasized in traditional computer science and software engineering curricula.

- focus on development *and acquisition*

  Many organizations acquire software from a variety of sources rather than internally developing software. The MSwA2010 curriculum must address the correct functionality and security of acquired as well as newly developed software. Although open source and COTS software are considered in some programs, acquisition processes are typically not a focus area for computer science and software engineering curricula.

- focus on *security* and correct functionality

  Assured software must not only be secure but provide correct functionality as well. Security must be an overarching focus area for the MSwA2010 curriculum; however, the curriculum must also address methods for assuring correct functionality. Some coverage of software security and technologies for verification and validation is often found in computer science and software engineering curricula; the MSwA2010 curriculum will apply and extend these foundations.

- focus on *software analytics*

  Assurance will often require analysis of existing software functionality and properties through reverse engineering methods. The MSwA2010 curriculum must address technologies for abstracting and assessing existing software and associated engineering artifacts. Computer science and software engineering curricula do not typically provide in-depth coverage of software analytics and reverse engineering topics.

- focus on *system operations*

  Assurance activities extend to monitoring system operations. The MSwA2010 curriculum must address monitoring technologies and methods, as well as recovery from intrusions and failures. While information technology (IT) and information systems programs often include

topics in system operations that can be built upon and extended, special emphasis on recovery and organizational continuity and survivability will be required as well.

- focus on *auditable evidence*

  Assurance activities must be guided by business environments, objectives, risks, and constraints and must produce auditable evidence for security properties and correct functionality that satisfies requirements, including compliance requirements. While some coverage of these topics can be found, they are not typically emphasized in traditional computer science and software engineering curricula.

These focus areas highlight unique properties of the MSwA2010 curriculum and also reveal areas of commonality and mutual reinforcement with computer science and software engineering programs. In some areas, the MSwA2010 curriculum builds on foundations from these curricula and extends coverage of technologies and processes for application to the specific needs of software assurance.

Although our focus on software assurance strongly emphasizes security and correct functionality, there are many other quality attributes that should be considered in software development, acquisition, and deployment. These include quality attributes such as performance, safety, modifiability, and privacy. In some cases, tradeoff analysis may be needed between some of these attributes, while in other cases the quality attributes may reinforce each other.

Let's take a closer look at safety as an example. Due to the increasing role of software in the nation's critical infrastructure, software's impact on system safety must be addressed. Examples of industrial control systems requiring particular attention are the power grid, nuclear power stations, water and food plants, chemical factories, oil refineries, railway systems, and air traffic control systems. Recently the tendency has been to replace older federated and well-protected discrete controls with new, integrated, complex digital systems that are not only interconnected in the control network but also connected to the general computing network—for the purpose of remote control, data collection, monitoring, and so on. Often developers of these new systems are not fully aware of the safety issues that such new architectures may bring, and IT professionals may neglect the need for additional safety precautions like analog or mechanical backup. Developers of control systems may also not be aware of security issues and vulnerabilities resulting from general computing network connectivity, and control engineers may not be familiar with operational security issues like leaving physical connections open, retaining default passwords, and not keeping anti-virus software up to date. To increase confidence in the assurance of industrial computer systems, security concerns have to be taken into account, and the mutual relationships of safety and security studied and reconciled.

Graduates of a Master of Software Assurance degree program must know how important the range of quality attributes are. As part of lifelong learning, they must also acquire the knowledge needed to address these and other quality attributes in specific domains and applications.

### Process Used to Develop MSwA2010 Curriculum Content

We used the following seven-step process to develop the software assurance curriculum topics, practices, knowledge units, outcomes, and core BoK.

### Step 1: Develop Project Guidelines

First, we developed a set of project guidelines, listed in Section 3. These guidelines provided foundation and guidance for the project scope, activities, and decision making. Their development also helped us to coalesce and better understand the project's purpose.

### Step 2: Identify and Review Sources

While addressing security during the software and system development life cycle is just starting to garner attention from project managers and business leaders, there is a growing body of knowledge on the subject. In parallel with the guidelines activity, we identified and reviewed credible and reputable sources of software security practices in industry, government, and academia (at the graduate and undergraduate levels). In all, 29 sources were considered; these are identified in Appendix B.

### Step 3: Define Topics

We used *Software Security Engineering: A Guide for Project Managers* as the organizing structure for our review of sources in Step 2 [Allen 2008]. The topics in this book apply to software assurance, even though the book is written for project managers. We supplemented the book's structure to reflect our experience, particularly with respect to software analysis, services, systems of systems, and technical issues that arose during our review. This resulted in identifying nine topics as follows. The tables in Appendix B include a mapping of knowledge areas to each topic.

1. Software security practices that span the SDLC (considered in all life-cycle phases)
2. Requirements engineering practices
3. Architecture and design practices
4. Coding practices
5. Testing practices
6. Analysis of software and services in static and operational contexts
7. Assembly, evolution, and deployment
8. Risk mitigation strategies for system complexity and scale
9. Governance and management practices

### Step 4: Define SDLC Practices and Categories

We evaluated sources for the nine topics listed above and detailed in Appendix B to identify and capture practices that fit within the scope of each topic. We included redundant practices to identify breadth of practice use across sources. Once all tables reflected practices from all applicable sources, we aggregated and abstracted the topics into high-level categories that could be used to describe groups of related practices. For example, in architecture and design, the categories are

- architecture
- design concepts
- module/component design
- detailed design
- design review and assessment

Governance and management, a broader practice topic, includes

- business case
- risk management
- awareness
- training
- project management
- software assurance practices integrated with the SDLC
- transition
- measurement
- ethics
- compliance
- evaluation
- acquisition

At the same time we were determining SDLC practices, we also developed the following four conceptual categories that helped us understand and assess the practices. The categories parse the definition of software assurance and also illuminate important dependencies among its elements. They serve as a high-level abstraction of the SDLC practices.

- security assurance

  Required levels of assurance cannot be achieved if security capabilities are insufficient for the threat environment. Security is the bedrock and centerpiece of the software assurance discipline.

- functionality assurance

  Defective software cannot be secure because defects can introduce vulnerabilities for attack. Security functionality must itself be properly implemented, or else it will fail to protect the software. The intended functionality of a system, that is, the services it provides for its users, must be properly implemented as well, or else it will fail to satisfy organizational objectives.

- operations assurance

  Security and functionality assurance are engineering activities that specify, develop, and evaluate system capabilities for dealing with threat environments while providing required services to users. These responsibilities extend to operations as well, to monitor and improve system capabilities in response to both changing threats and evolving user needs. In addition, the assurance discipline must provide operational means to respond to intrusions and maintain continuity of operations in adverse circumstances.

- assurance processes and management

  The activities of security, functionality, and operations assurance can involve many tasks and participants over substantial periods of time. To be effective, these activities require processes and practices within the context of organizational objectives and constraints, as well as planning, scheduling, tracking, and reporting in their execution.

The relationship of these conceptual categories to the overall MSwA2010 content development process is depicted in Figure 1 and Figure 2.

### Step 5: Solicit External Feedback

Once we defined SDLC practices and categories (see Appendix B), we sought input from representatives among managers, practitioners, and educators. We wanted to know their requirements for graduates of degree programs based on this curriculum and insights on curriculum outcomes. We also wanted to make sure we were on the right track and had not overlooked a significant reference, source, or curriculum. We developed a three-page questionnaire, included in Appendix C. The questionnaire sought answers to these two high-level questions:

- Assume you are interviewing to fill a position for a specialist in software assurance or software security. Please rate each of the following capabilities[6] you might consider in hiring such an individual. Provide responses using two ways of rating these capabilities:
  - how you currently rate such capabilities for a prospective employee regardless of their academic background and experience (current)
  - how you would rate a prospective employee who had a master's degree with a focus on software assurance
- Are there other capabilities or issues that are important to your organization when hiring a software assurance or software security professional?

Responses to our questionnaire are summarized in Appendix C. As expected, these responses resulted in updates to the practices tables in Appendix B.

### Step 6: Develop Outcomes and Core Body of Knowledge

While we were capturing external feedback, we were also working on identifying curriculum outcomes (refer to Section 4) and the core BoK (refer to Section 7).

Figure 1 is a high-level view of the process we used to develop the outcomes and BoK. As indicated in the figure, the project guidelines greatly influenced all subsequent project activities, including developing the outcomes and BoK.

The outcomes were also significantly influenced by the GSwE2009 and questionnaire responses. Outcomes were essentially a refinement and evolution of the SDLC practices and categories (Step 4) and represent the essence of what should be expected from a graduate of an MSwA degree program. Refer to Section 4 for a description of the MSwA2010 outcomes.

[6] Examples include: think like an attacker; apply software assurance practices during requirements engineering, architecture and design, coding, and testing; be able to make technical arguments on the value of software assurance.

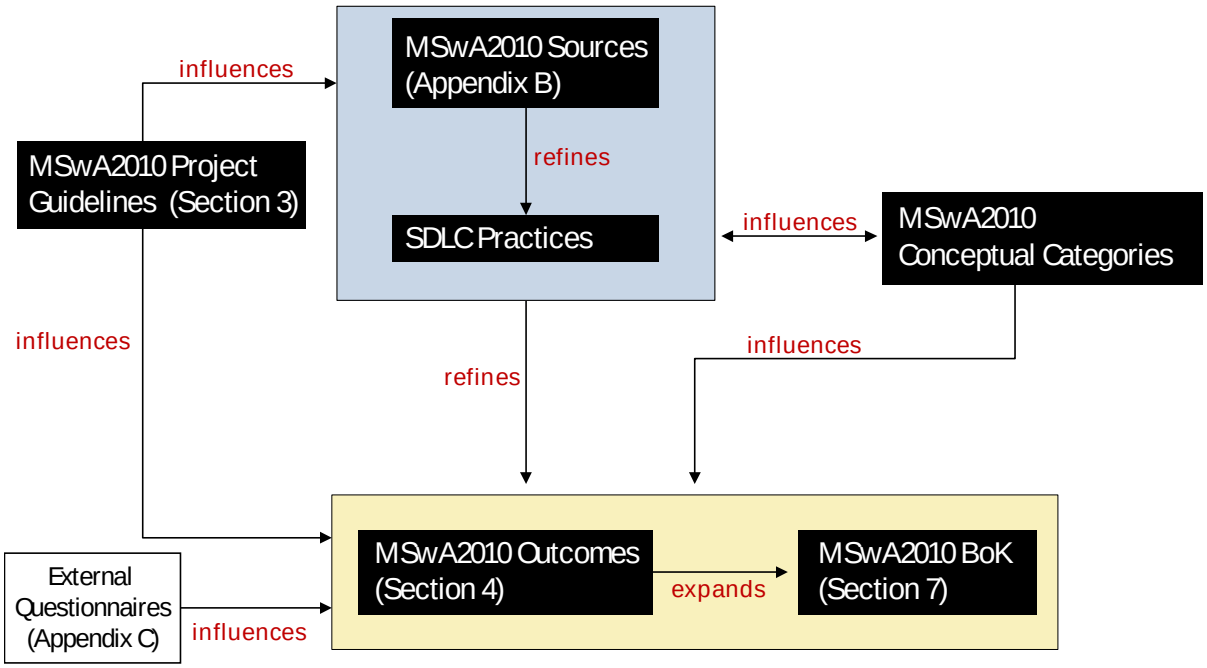


*Figure 1: High-Level View of the MSwA2010 Project Process*

We developed the BoK by adding detail and expanding each outcome in a three-level structure: a knowledge area expanded into knowledge units, and each knowledge unit expanded into knowledge topics. Refer to Section 7 for a description of the knowledge areas, units, and topics.

We used the project guidelines and conceptual components as a guidance framework to develop both the outcomes and BoK. The external questionnaire (Step 5) was an important influence on this activity by providing a check on the relevance and currency of the process used to develop the MSwA2010 curriculum content.

Figure 2 is an expansion of Figure 1, providing additional detail on the MSwA2010 content development process.

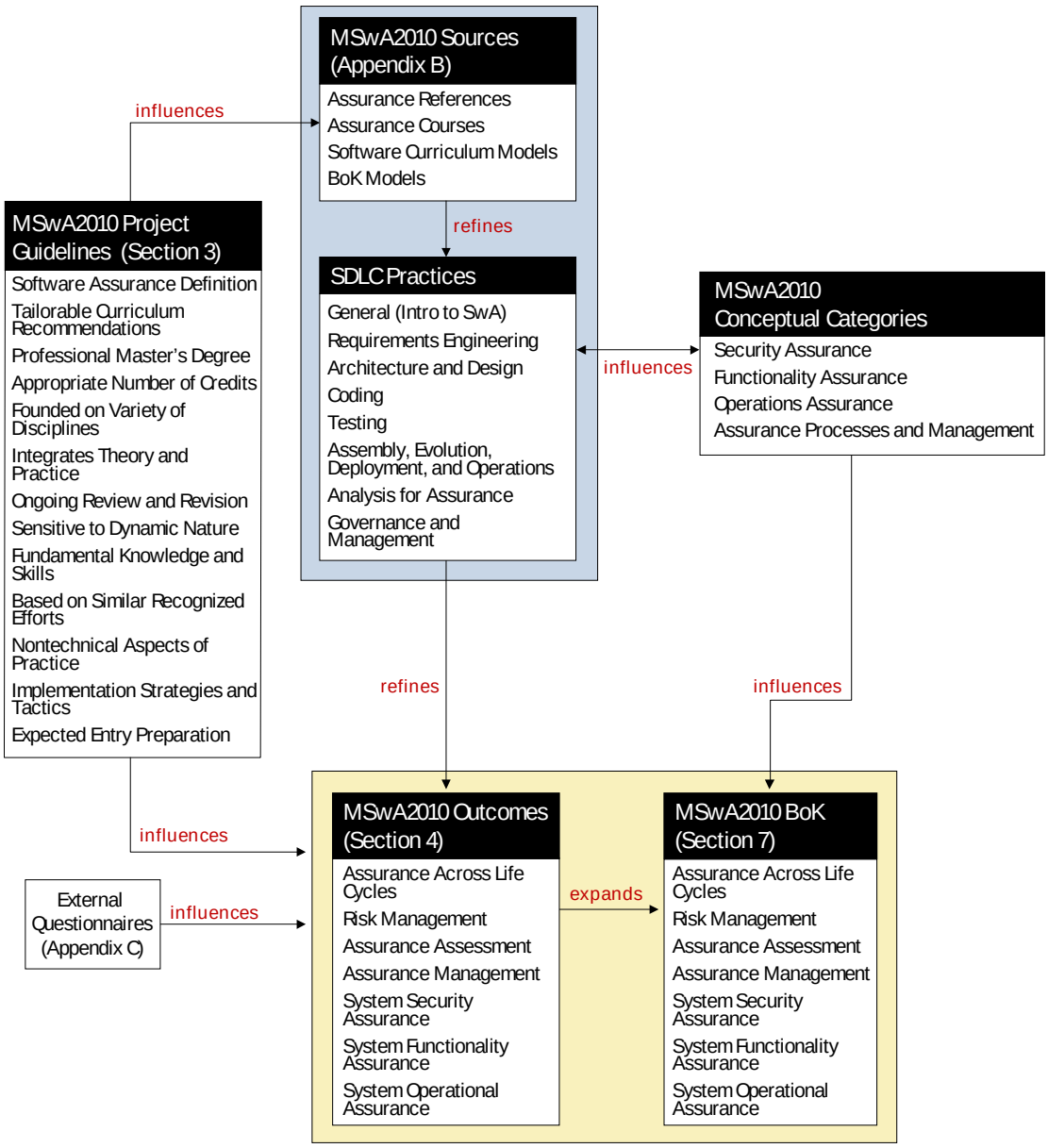


*Figure 2: Detailed View of the MSwA2010 Project Process*

### Step 7: Compare Knowledge Units to Practices

We performed a cursory gap analysis by comparing the BoK knowledge units to the SDLC practices and categories. We wanted to ensure that all practice categories were covered by at least one knowledge unit or that we made a conscious decision to exclude some practice topic, for example, privacy, if we felt it was out of scope. The results of this effort appear in Appendix B. This exercise was neither exhaustive nor rigorous, and the result should not be construed as a complete traceability.

This cross-check did result in updates to the practices tables, knowledge units, and outcomes, so it accomplished the desired result.

In the next section, we present MSwA2010 project guidelines that we used to establish the foundation, scope, and boundaries for project activities and decision making.

# 3 Guidelines for Developing This Curriculum

Our first task was to develop guidelines that we would follow for the MSwA2010 curriculum project. The guidance from the GSwE2009 document, with some modifications, served our purpose.

We adapted the first half of the guidelines to describe general characteristics of the MSwA2010 curriculum development process.

1. For purposes of this report, we define software assurance as

   *Application of technologies and processes to achieve a required level of confidence*[7] *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.* (Refer to Section 2 for a detailed discussion of this definition.)
2. The principal purpose of the MSwA2010 project will be to provide a set of customizable recommendations for developing and improving curricula that provide software assurance education at the master's degree level. Although it is not intended to be the basis for accreditation (or certification), it may be useful as a reference for curriculum assessment.
3. The master's degree described by the MSwA2010 project will be a professional degree targeting software assurance practitioners. With modification, MSwA2010 may serve as the foundation for those with a research interest who ultimately seek a doctoral degree; however, MSwA2010 is designed specifically to support professional degrees.
4. A master's program that satisfies MSwA2010 should require about as many credits as typical U.S. programs do now.[8]
5. Software assurance draws its foundations from a wide variety of disciplines.
6. All software assurance students must learn to integrate theory and practice.
7. The rapid evolution and the professional nature of software assurance require an ongoing review and revision of the corresponding curriculum.

The second half of the guidelines is more prescriptive and tactical.

[7] In the CNSS definition, the use of the word "confidence" implies that there is a basis for the belief that software systems and services function in the intended manner.

[8] There is a difference between U.S. and other educational systems in assigning credits. Typically, a U.S. program requires a certain number of credits for graduation, awarded by taking courses, each of which has an associated number of credits. Historically, the number of credits per course has often aligned with the number of hours of lecture per week, but with online and other non-traditional formats becoming increasingly popular, the rules for assigning credits to a class have become more varied. In Europe, the European Credit Transfer and Accumulation System (ECTS) is a credit system introduced in 1989 that has been successfully tested and used across Europe. ECTS is based on the student workload required to achieve the objectives of a program, objectives preferably specified in terms of the learning outcomes and competences to be acquired. ECTS was set up initially for credit transfer. The system facilitated the recognition of periods of study abroad and thus enhanced the quality and volume of student mobility in Europe. Recently ECTS is developing into an accumulation system to be implemented at institutional, regional, national, and European levels.

8. MSwA2010 will be sensitive to the fact that there will be changes in technologies, practices, applications, and new developments in pedagogy and will stress the importance of lifelong learning, even though we are not able to anticipate these changes in this initial report.
9. MSwA2010 will identify the fundamental skills and knowledge that all graduates of an SwA master's degree program should possess.
10. MSwA2010 will be based on a flexible curriculum architecture and on recognized bodies of knowledge, such as the Software Engineering Body of Knowledge (SWEBOK) [IEEE-CS 2004c] and the Software Assurance Common Body of Knowledge (SwACBK) [DHS 2010b]. Other recognized bodies of knowledge that are referenced include the GSwE2009, the earlier MSE model curricula [Ardis 1989, Ford 1991], and other relevant documents.
11. MSwA2010 will honor individual program and student flexibility by limiting the common knowledge required for all students to no more than 50 percent of the total knowledge taught in a master's program.
12. MSwA2010 will include exposure to nontechnical aspects of professional practice as an integral component of the graduate curriculum, such as ethics and teamwork.
13. MSwA2010 will include discussions of strategies and tactics for implementation of the curriculum, along with high-level recommendations.
14. MSwA2010 will identify expected knowledge and experience for students to enter a master's program in software assurance (refer to Section 5).

These project guidelines significantly influenced the development of curriculum outcomes described in the next section. In addition, the outcomes were influenced by the GSwE2009 and responses to our external questionnaire (Section 2, Step 5). Outcomes were essentially a refinement and evolution of the SDLC practices and categories (Section 2, Step 4) and represent the essence of what should be expected from a graduate of an MSwA2010 program.

# 4 Proposed Outcomes When a Student Graduates

The outcomes described in this section specify the knowledge, skills, and capabilities that graduates of an MSwA program should have when they complete the program. The outcomes represent the minimum capabilities that should be expected of a professional in the area of software assurance when they complete an MSwA program.

The primary audience for the MSwA2010 project, the graduate faculty, should be prepared to teach courses that achieve these outcomes. Software development and acquisition employers responsible for staffing software assurance positions and developing increased software assurance capabilities of their current employees should expect that graduates of an MSwA program be proficient in capabilities described in these outcomes. The outcomes also provide a model for curriculum content, organization, and support to those who assess software assurance programs. The outcomes can be grouped in two main areas: (1) assurance process and management and (2) assurance product and technology.

## Assurance Process and Management

### Outcome 1. Assurance Across Life Cycles

#### Brief description

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.

#### Detailed description

Threats, attacks, and vulnerabilities must be considered whether a software system or component is developed from inception, obtained through reuse or acquisition, modified through maintenance, or replaced with new versions. To specify, design, build, acquire, deploy, and operate software that minimizes vulnerabilities and isolates or limits the effects of threats and attacks, software security activities and practices must be integrated throughout the software life-cycle process and adapted to current software engineering practices and methodologies.

Program graduates will understand and be able to judge which software security methods, techniques, and tools are needed in the development phases of requirements analysis and specification, architectural and component design, unit implementation, assembly and integration, and review, testing, and evaluation. Graduates will be able to assess security concerns and determine appropriate processes and models to address such concerns in the

- development and deployment of software systems
- operational environment of software
- evolution of a software system
- acquisition of commercial and open-source software
- monitoring of ongoing security support services from software suppliers

Graduates will be able to define security requirements in situations where the software itself is not available for analysis. They will also be able to monitor and assess the security performance of services in operational use.

### Outcome 2. Risk Management[9]

#### Brief description

Graduates will have the ability to perform risk analysis and tradeoff assessment, and to prioritize security measures.

#### Detailed description

A software security risk exists when a particular threat can exploit a vulnerability that can have harmful effects in a system. Such risks must be managed in order to ensure that software is resilient and resistant to threats.

Graduates of this program will have knowledge of and experience with using software security risk management techniques, methods, and practices. They will be able to perform risk assessment by analyzing, identifying, and modeling potential threats and vulnerabilities and ranking them according to the likelihood of exploitation and to severity and magnitude of impact. Graduates will be able to perform cost-benefit analysis within project constraints to assure correct software system functionality, achieve assurance objectives, and assess the possible presence of malicious content or corrupted functions.

Graduates will be able to identify and analyze the changes to software requirements and software design, code, deployment, and operational procedures that are needed to eliminate vulnerabilities and mitigate threats. They will be able to estimate the costs of such changes and the costs of the impact of the associated risks, and perform a tradeoff analysis to guide the selection of security technologies and methods to manage and mitigate risks.

### Outcome 3. Assurance Assessment

#### Brief description

Graduates will have the ability to analyze and validate the effectiveness of assurance operations and create auditable evidence of security measures.

#### Detailed description

To analyze the effectiveness of assurance technologies and methods, graduates will be able to answer the following questions:

1. How do I establish and specify the required or desired level of assurance for a specific software application, set of applications, or a software-intensive system?
2. How do I measure, at each phase of the development or acquisition life cycle, that the required or desired level of assurance has been achieved?

[9] This refers to system or software risk management, not organizational risk management.

In response to the first question, graduates will be able to define and develop a baseline against which software assurance can be measured. Methods may include

- verifying and validating that security requirements have been satisfied throughout the SDLC (refer to Outcome 1)
- using a range of risk analysis approaches, including being able to prioritize software components and systems based on their contribution to mission success (higher priority components require greater levels of assurance) (refer to Outcome 2)
- developing and using auditable assurance evidence throughout the SDLC (refer to Outcome 6)

Regarding the second question, graduates will be able to define and develop key product measurements, process measurements, and other performance indicators that can be used to validate the required level of software assurance appropriate to a given life-cycle phase. Graduates will be able to articulate and use a software assurance measurement process and framework that can be tailored for a specific development project.

Graduates will be able to define a required level of assurance for all life-cycle phases of an in-class development project and present measurements that demonstrate whether the required level has been satisfactorily achieved.

### Outcome 4. Assurance Management

#### Brief description

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.

#### Detailed description

Graduates will be able to communicate compelling business and technical arguments on the value of software assurance to executives, project managers, and peers to catalyze adoption of assurance practices. To make the business case for software assurance, graduates will be able to formulate and present economic and other arguments that describe the need for software assurance and the impact if software assurance is not addressed during software development and acquisition. Arguments may include compliance with legal, regulatory, and standards-based requirements; ensuring continuity of operations; cost-benefit models; risk impact; cost and loss avoidance; and methods and analytical tools for return on investment.

In developing such arguments, graduates will be able to determine if development and operations life-cycle costs for incorporating a required level of software assurance are consistent with business needs. Graduates will be able to demonstrate that the required investment in cost, staff resources, schedule, and other forms of investment are commensurate with the value of the software and system.

Graduates will understand how to lead software and system assurance efforts (as an extension of program and project management skills). Graduates will also understand the importance of staying current with changing and emerging security trends, technologies, and methods.

## Assurance Product and Technology

### Outcome 5. System Security Assurance

#### Brief description

Graduates will have the ability to incorporate effective security technologies and methods into new and existing systems.

#### Detailed description

Graduates will be able to understand the concepts and operations of new and existing diverse systems.[10] They will be able to identify the threats caused by both malicious acts and accidental events such as failures related to COTS and open-source proliferation, internet connectivity, and the wireless web. Critical infrastructure industrial systems, like energy systems, should be an important target for assurance activities.

Graduates will understand the process of implementing cybersecurity into new and existing systems (including availability, integrity, privacy, confidentiality, and non-repudiation) using identification, authorization, and authentication concepts. They will understand how to participate in the development and acquisition of diverse systems that have appropriate security capabilities.

Graduates will be familiar with appropriate countermeasures, such as network defense of industrial control systems in the areas of

- design and planning—for example, layers, access control, privileges
- technology—for example, firewalls, intrusion detection, virus control, encryption
- people and policies—for example, procedures, standards, documentation, training, audits, checklists
- physical and personnel security—for example, gates, locks, guards, ID cards

Graduates will acquire the ability to think like an attacker in evaluating threat environments, system vulnerabilities, and security properties. In becoming familiar with secure coding methods and templates to use in new system development, they will be able to assess systems for the presence of vulnerabilities and provide coding solutions to minimize or eliminate them.

To analyze the threat environment, graduates must be able to understand and duplicate the techniques that have been used by attackers to interfere with an application's or a system's operations.

Graduates will understand how to execute a variety of attacks, including password cracking, escalation of privileges, denial-of-service, and the creation, distribution, and insertion of viruses, worms, Trojans, spyware, and logic bombs. Attacks also include those that take advantage of buffer overflows, cross-site scripting, SQL injections, IP and server spoofing, and session hijacking. Graduates must learn about these techniques and agree that they will not use these techniques for the purpose of attacking others. Of course, in the classroom, there is no way to prevent unethical behavior, but we attempt to address this issue in the curriculum (see Outcome

[10] See the Glossary for a definition of diverse systems.

5.3 in Section 7). Graduates will also be able to determine which of these methods an attacker is most likely to use to accomplish specific objectives.

### Outcome 6. System Functionality Assurance

#### Brief description

Graduates will have the ability to verify new and existing software system functionality for conformance to requirements and to help reveal malicious content.

#### Detailed description

Graduates will have the ability to assure developed and acquired system and service security and functionality. They will gain expertise in developing and assessing requirements, specifications, designs, and implementations; performing verification and testing; and analyzing existing systems through reverse engineering techniques.

- analyzing and evaluating development processes, environments, and technologies
- evaluating requirements for completeness and correctness
- applying rigorous methods for software specification, design, implementation, correctness verification, and testing
- applying software quality and process engineering methods
- structuring unstructured software for improved understanding and analysis
- reverse engineering software to determine functional behavior and reveal vulnerabilities or malicious content
- using automated tools to analyze software properties
- developing auditable assurance evidence
- analyzing the assurance of open source, COTS, and government, off-the-shelf (GOTS) software
- analyzing test and evaluation processes for assuring software systems
- defining assurance requirements across supply chains and assurance assessment of acquired software (refer to Outcome 3)
- creating service agreements that define required functionality and levels of service from providers

### Outcome 7. System Operational Assurance

#### Brief description

Graduates will have the ability to monitor and assess system operational security and respond to new threats.

#### Detailed description

For all classes of software and systems (new, existing, acquired, diverse, etc.), it is essential for graduates to be able to monitor and assess if software and systems are operating securely and functioning as intended (see also Outcomes 3 and 6). Software and systems need to resist, respond to, and recover from threats and attacks that were considered during development (for new

systems) and evaluated (for other classes of systems) before being placed into an operational environment. Graduates must demonstrate this same ability for new threats and be able to identify corresponding shortfalls in security function and performance and identification of relevant countermeasures to address these shortfalls. Monitoring and assessing system operational security includes identifying vulnerabilities and other issues that could have been addressed earlier in the development or acquisition life cycle and ensuring that those issues are submitted as potential process improvements.

Specific capabilities include

- implementing system, service, and personnel monitors and controls
- monitoring and controlling system and service operations for security and correct functionality
- developing procedures and training for system users and system administrators
- performing analysis of malware and developing countermeasures
- effectively responding to accidents, failures, and intrusions
- assuring business survivability and operational continuity
- acquiring systems and services with appropriate security capabilities
- defining capabilities and limitations and applying operational monitoring automation

Graduates will be able to use a range of technologies and methods to monitor and assess operational software and systems to determine that they continue to meet their security requirements, perform as expected, and continue to provide critical services in the face of known and new threats. Several of these technologies and methods are described in Outcomes 3 and 6.

The next section describes required and desired prerequisite knowledge and skills that students of an MSwA program should have mastered prior to starting this curriculum. Gaps in satisfying prerequisites may put new students at a disadvantage, perhaps requiring remedial work for successful completion. Faculty members offering this curriculum should be prepared to deal with students' knowledge and skill gaps and offer possible solutions.

# 5 Background Expected of Students Entering the Program (Prerequisites)

As with all scientific and engineering disciplines, appropriate prerequisite knowledge and skills are essential for successful graduate-level education in software assurance. Because an MSwA program is an extension and specialization of undergraduate education, it is important to define required foundations upon which to build professional capabilities. This report defines prerequisite foundations from a security perspective, leaving areas such as reliability and safety for other analyses. Some of these foundations are necessarily general in nature; others are more specific to security aspects of software assurance. Because of the technical nature of software assurance, we anticipate that entrants to a program based on MSwA2010 will hold undergraduate degrees in disciplines such as computer science; software engineering; electrical, electronic, and computer engineering; mathematics; or information systems. We expect that prerequisites will be satisfied through some combination of undergraduate courses, work experience, and possibly remedial education prior to the start of an MSwA program.

The following prerequisites are organized into three categories: computing foundations, software engineering, and security engineering. While satisfying substantially all prerequisites is an ideal goal, it is expected as a practical matter that candidates will satisfy the majority of the computing foundations prerequisites, plus elements of either the software engineering or security engineering prerequisites. Candidates who have been in the workforce for a while may have work experience that satisfies the prerequisites rather than formal coursework. Alternatively the university may use exams to allow a candidate to "test out" of a prerequisite. In some areas there may be certification programs, such as the IEEE Certified Software Development Professional (CSDP) program[11] that can be substituted for coursework. To help set expectations, these prerequisites are defined in terms of the following Bloom cognitive levels, which are described in Appendix A:

- knowledge (K)
- comprehension (C)
- application (AP)
- analysis (AN)

## Computing Foundations

### Discrete Mathematics

Justification

- The ability to work with discrete mathematics provides the foundation for key software assurance technologies and methods taught in an MSwA program.

Prerequisites

- sets, functions, and relations; graphs and trees; propositional and predicate logic; number systems; modular arithmetic; proof techniques (Bloom Level C)

[11] See http://www.computer.org/portal/web/certification/csdp for more information.

- fundamentals of probability and statistics (Bloom Level C)

Source

- undergraduate discrete mathematics, probability, and statistics courses[12]

### Computer Fundamentals

Justification

- Understanding computer hardware organization provides a foundation for implementation-level assurance technologies and methods taught in an MSwA program.

Prerequisites

- computer hardware function and organization, assembly and microcode organization, memory organization and access, communication interfaces, multiprocessing (Bloom Level C)

Source

- undergraduate introduction to computing and computer architecture courses

### Networks and Communications

Justification

- Understanding network architectures provides a foundation for technologies and methods of assurance for distributed computing and large-scale systems taught in an MSwA program.

Prerequisites

- network-centric computing and communication, network topologies and protocols, addressing and routing (Bloom Level C)
- web applications and multimedia, wireless and mobile computing (Bloom Level C)
- network configuration, monitoring, performance, and management (Bloom Level C)

Source

- undergraduate networking and communication course

### Programming Environments

Justification

- The ability to work with programming environments provides a foundation for implementation-level assurance technologies and methods taught in an MSwA program.

Prerequisites

- operating systems, including scheduling, memory management, concurrency, security, file system, and device management (Bloom Level C)
- real-time and embedded system concepts (Bloom Level C)

[12] Note that candidates who have been in the workforce for a while may not have this background. They should be able to acquire the background through self-study or in the MSwA program itself.

- implementation environments, including programming languages, compilers, assemblers, loaders, and libraries (Bloom Level C)
- support tools for analysis of programs and systems (Bloom Level C)

Source

- undergraduate operating system and programming courses[13]

### Program Development

Justification

- The ability to write programs using contemporary languages is a fundamental skill for many subject areas taught in an MSwA program.

Prerequisites

- object-oriented programming using a language such as Java or C++ (Bloom Level AP)
- input/output streams and process threads, pointers, memory allocation and deallocation (Bloom Level AP)
- fundamental data structures including arrays, lists, queues, and stacks (Bloom Level AP)
- analysis and implementation of basic algorithms (Bloom Level AP)
- semantic foundations of programming languages (Bloom Level K)

Source

- undergraduate programming[14] and fundamentals of algorithms courses

## Software Engineering

### Software Development Life Cycle

Justification

- Understanding life-cycle stages provides a framework for application of assurance technologies and methods taught in an MSwA program.

Prerequisites

- requirements elicitation, analysis, and validation; use case and flow analysis; system and software specification; architecture, design, and implementation; verification, inspection, and testing; and evolution (Bloom Level C)
- software quality, dependability, and reliability (Bloom Level C)
- project management concepts, including work breakdown, scheduling, budgeting, tracking, and risk management (Bloom Level C)
- system development models, including incremental, spiral, and agile methods; project management frameworks including the Capability Maturity Model Integration (CMMI®[15]) framework (Bloom Level C)

---

[13] Note that programming experience may serve as a substitute for a programming course.

[14] Note that programming experience may serve as a substitute for a programming course.

[15] ® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Source

- undergraduate software engineering and systems design courses

### Software Analysis

Justification

- Awareness of software analysis methods provides a foundation for understanding technologies for reverse engineering and analysis taught in an MSwA program.

Prerequisites

- software analysis tools and methods, including static and dynamic analyzers (Bloom Level C)
- reverse engineering methods, including transformation of unstructured code into structured form and analysis of program behavior (Bloom Level C)

Source

- undergraduate software engineering and systems analysis courses

## Security Engineering

### Security Issues

Justification

- Awareness of security issues and requirements is fundamental background for pursuit of an MSwA degree.

Prerequisites

- knowledge of security threats from criminal, nation-state, and insider adversaries; consequences of attacks on critical infrastructure, defense, and economic systems; security risks and requirements for application domains such as finance, energy, and transportation (Bloom Level C)
- security issues in computing trends, including global networks, systems-of-systems, open-source, cloud computing, cross-site scripting, web security, and social networking (Bloom Level C)
- security properties, including privacy, confidentiality, authentication, authorization, availability, integrity, and non-repudiation (Bloom Level C)
- security aspects of human behavior in interacting with software systems (Bloom Level C)

Source

- undergraduate computer security course

The next section presents candidate architectures and course packages that can be used to organize the MSwA2010 BoK to achieve the outcomes described in Section 4. The architecture provides for preparatory content, core course content, elective content, and a capstone experience through which students can demonstrate their understanding and ability to apply what they have learned.

# 6 MSwA2010 Curriculum Architecture

This section describes a curriculum architecture and course packages that can be used to organize and package the body of knowledge that makes up the MSwA2010 curriculum. Using the MSwA2010 BoK, the architecture in this section identifies the minimum content that all degree programs should include. The curriculum architecture is similar to the one proposed in the GSwE2009 and is compatible with software engineering master's programs that are based on that curriculum. It is intended to provide a structural basis for programs that deliver the outcomes described in Section 4.

### MSwA Degree Program

The curriculum architecture includes preparatory material, core materials, elective materials, and a capstone experience.

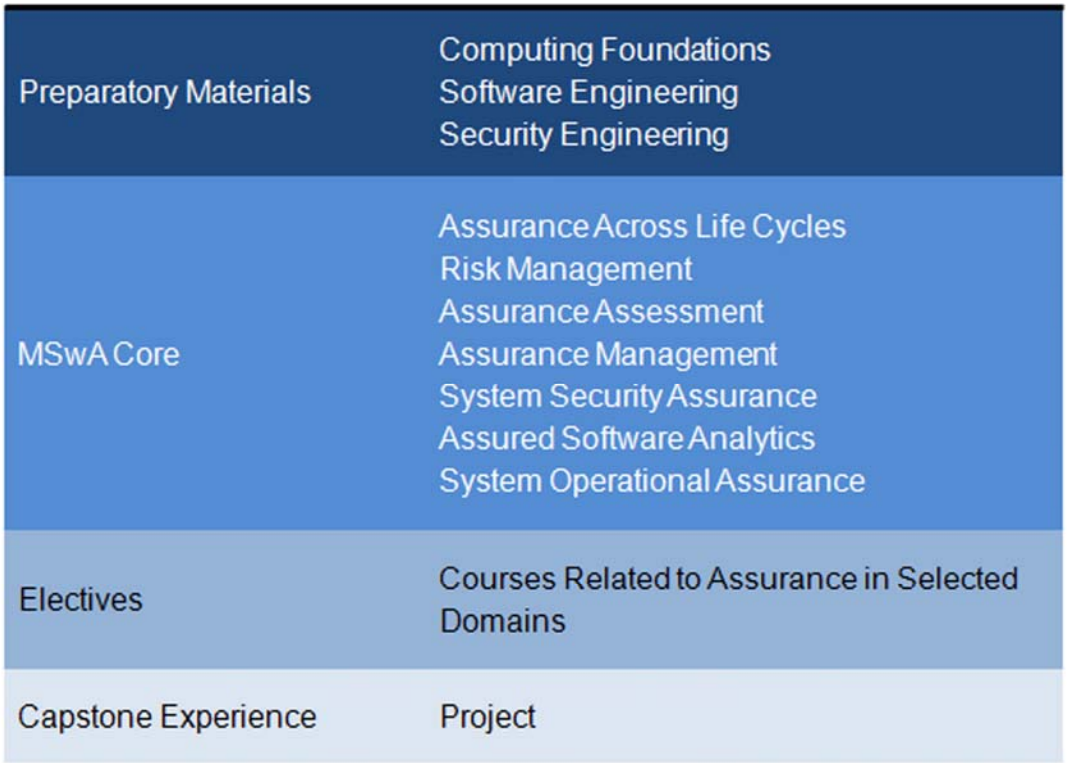


*Figure 3: Architecture of an MSwA Degree Program*

Figure 3 provides an overview of the MSwA2010 degree program curriculum architecture. The Preparatory Materials represent the prerequisites described in Section 5.This material should be mastered by students before entering the master's program. The MSwA Core material and everything below it is mastered after program entry. Individual programs will determine how to prepare students whose background falls short. Typically, colleges and universities that wish to admit students who lack the expected background will provide preparatory courses that those

students should take before entering the master's program. The more deficient the students' backgrounds are relative to the prerequisites, the higher the risk that students will not perform satisfactorily, harming themselves and fellow students.

The MSwA Core includes the fundamental skills and knowledge all MSwA graduates should have; these skills are detailed in the MSwA BoK (refer to Section 7). Where appropriate, the MSwA Core emphasizes the guidelines used to define the MSwA2010 BoK, including its dependencies on other related disciplines, such as software engineering, testing, and project management; all graduate programs should include this material. Courses that teach core content are mandatory.

Electives accommodate individual students' interests and may cover unique requirements of a program or institution. Students may take electives to gain more depth in a core area (for example, assurance assessment) or to extend and broaden their knowledge in a particular application domain (for example, application to a particular market sector). Because software assurance is a relatively new academic field, it is likely that special topics courses and seminars will be included among the electives.

The MSwA2010 project recommends that students demonstrate their accumulated skills and knowledge in a capstone experience, which engages the student in a realistic team project emphasizing software assurance concepts and practices. The capstone experience would likely be between three and six semester credit hours, which would count toward the total credit hours typically required for a master's degree. In this context, a capstone project would ideally be a practical software assurance undertaking, using best software assurance practices and tools with a real customer that has actual software assurance objectives. Students completing the curriculum must be able to understand and appreciate the skills needed to produce assured software in a typical software development environment. These topics should be integrated into the core materials and perhaps could be reinforced in the elective materials. However, the presence of a capstone project is of considerable importance, as it offers students the opportunity to tackle a major project that is likely to be more comprehensive in gaining realistic software assurance experience than their prior projects.

This architecture does not imply that there are courses with names corresponding to the curriculum knowledge areas. Sample course descriptions are provided in Appendix F, but the architecture is intended to be independent of actual course composition. Figure 4 provides an example of typical course packaging. In this example, Courses 1 and 2 include only core material or elective material, respectively, whereas Course 3 covers a combination of core and elective materials. This architecture also does not imply specific course sequencing. Courses containing preparatory or core materials need not be completed before coursework in the next row can begin. Sequencing of courses should be tailored to a specific institution's curriculum.

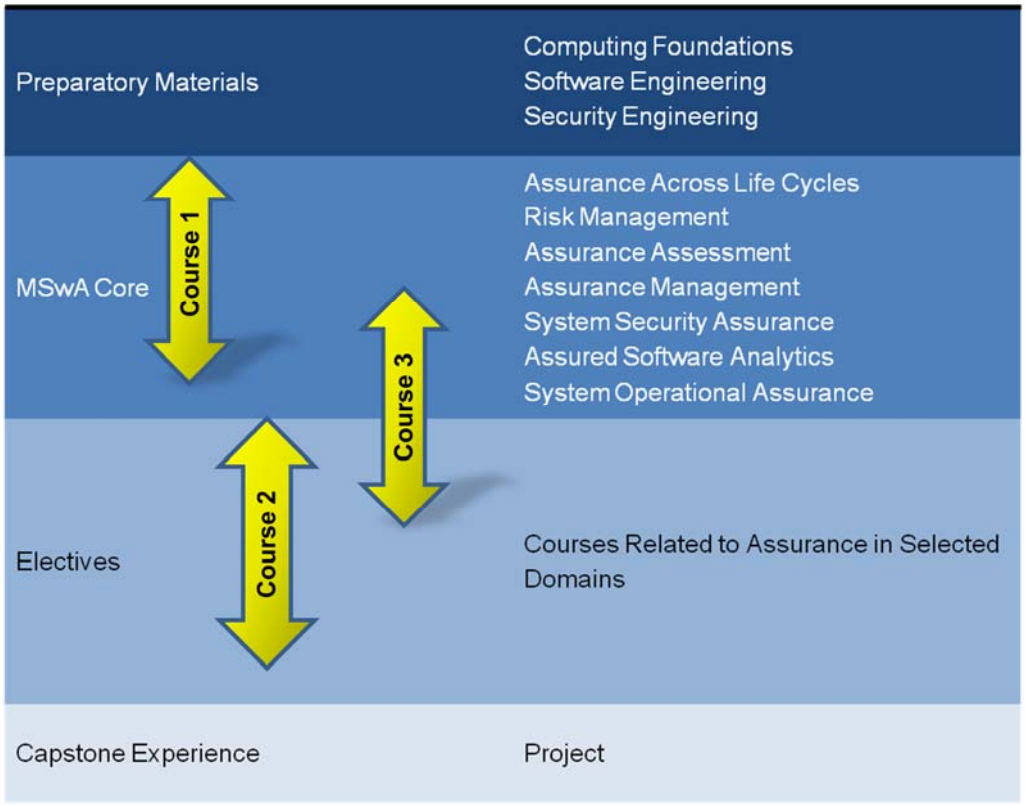


*Figure 4: Course Alignment Across MSwA Core and Electives*

### MSE Degree Program with Software Assurance Track

Figure 5 illustrates a complete program that includes Preparatory Materials, the GSwE Core, the MSwA Core, and the Capstone Experience. Note that Software Quality in the GSwE Core is defined as the capability of a software product to satisfy stated and implied needs when used under specified conditions. Graduates of such a program would have a Master of Software Engineering with a Software Assurance specialization that includes the entire MSwA core. Note that the items in the right column correspond to knowledge areas, not courses, so the number of items listed under GSwE core and MSwA core is not related to specific courses or credit hours in these areas. As shown, such a program would leave little or no room for electives.

| | |
|---|---|
| Preparatory Materials | Computing Foundations<br>Software Engineering<br>Security Engineering |
| GSwE Core | Ethics and Professional Conduct<br>Systems Engineering<br>Requirements Engineering<br>Software Design<br>Software Construction<br>Software Testing<br>Software Maintenance<br>Configuration Management<br>Software Engineering Management<br>Software Engineering Processes<br>Software Quality |
| MSwA Core | Assurance Across Life Cycles<br>Risk Management<br>Assurance Assessment<br>Assurance Management<br>System Security Assurance<br>Assured Software Analytics<br>System Operational Assurance |
| Capstone Experience | Project |

*Figure 5: MSE with SwA Specialization*

The MSwA2010 curriculum architecture provides the organizing structure for the core BoK in the next section. The core BoK derives from all work described up to this point in this report, as shown in Figure 1 and Figure 2. The BoK is structured to aid faculty members in selecting and developing course content that fits with their programs and meets their objectives. The BoK directly supports the achievement of the outcomes described in Section 4 and will provide graduates with fundamental knowledge, skills, and capabilities in software assurance.

# 7 Core Body of Knowledge

This section describes the MSwA2010 BoK, the core body of knowledge for an MSwA degree. The term *software assurance* used in this section is the expanded definition in Section 2 of this report. The MSwA2010 BoK includes software assurance practices that are required to support the MSwA2010 outcomes. All software assurance professionals must know these practices to perform their jobs effectively. The MSwA2010 BoK is structured into seven knowledge areas, with each knowledge area subdivided into a set of knowledge units.

The MSwA2010 BoK does not provide detailed descriptions but rather serves as a guide to the body of knowledge by referencing literature that explains and elaborates on the elements (see Appendix B).

The following knowledge areas are defined in terms of the Bloom cognitive levels, which are described in Appendix A. Brief descriptions of the outcomes are included for each knowledge area. For detailed descriptions of the outcomes, refer to Section 4.

### 1. Assurance Across Life Cycles

Outcome: 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.

1.1. Software Life-Cycle Processes

1.1.1. New development (Bloom Level C)

Processes associated with the full development of a software system

1.1.2. Integration, assembly, and deployment (Bloom Level C)

Processes concerned with the final phases of the development of a new or modified software system

1.1.3. Operation and evolution (Bloom Level C)

Processes that guide the operation of the software product and its change over time

1.1.4. Acquisition, supply, and service (Bloom Level C)

Processes that support acquisition, supply, or service of a software system

1.2. Software Assurance Processes and Practices

1.2.1. Process and practice assessment (Bloom Level AP)

Methods, procedures, and tools used to assess assurance processes and practices

1.2.2. Software assurance integration into SDLC phases (Bloom Level AP)

Integration of assurance practices into typical life-cycle phases (for example, requirements engineering, architecture and design, coding, test, evolution, acquisition, and retirement)

## 2. Risk Management

Outcome: Graduates will have the ability to perform risk analysis and tradeoff assessment, and to prioritize security measures.

2.1. Risk Management Concepts

2.1.1. Types and classification (Bloom Level C)

Different classes of risks (for example, business, project, technical)

2.1.2. Probability, impact, severity (Bloom Level C)

Basic elements of risk analysis

2.1.3. Models, processes, metrics (Bloom Level C)

Models, process, and metrics used in risk management

2.2. Risk Management Process

2.2.1. Identification (Bloom Level AP)

Identification and classification of risks associated with a project

2.2.2. Analysis (Bloom Level AP)

Analysis of the likelihood, impact, and severity of each identified risk

2.2.3. Planning (Bloom Level AP)

Risk management plan covering risk avoidance and mitigation

2.2.4. Monitoring and management (Bloom Level AP)

Assessment and monitoring of risk occurrence and management of risk mitigation

2.3. Software Assurance Risk Management

2.3.1. Vulnerability and threat identification (Bloom Level AP)

Application of risk analysis techniques to vulnerability and threat risks

2.3.2. Analysis of software assurance risks (Bloom Level AP)

Analysis of risks for both new and existing systems

2.3.3. Software assurance risk mitigation (Bloom Level AP)

Plan for and mitigation of software assurance risks

2.3.4. Assessment of Software Assurance Processes and Practices (Bloom Level AP)

As part of risk avoidance and mitigation, assessment of the identification and use of appropriate software assurance processes and practices

## 3. Assurance Assessment

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

3.1. Assurance Assessment Concepts

3.1.1. Baseline level of assurance; allowable tolerances, if quantitative (Bloom Level AP)

Establishment and specification of the required or desired level of assurance for a specific software application, set of applications, or software-reliant system (and tolerance for same)

3.1.2. Assessment methods (Bloom Level C)

Knowledge of how various methods (such as validation of security requirements, risk analysis, threat analysis, vulnerability assessments and scans, and assurance evidence) can be used to determine if the software/system being assessed is sufficiently secure within tolerances

3.2. Measurement for Assessing Assurance

3.2.1. Product and process measures by life-cycle phase (Bloom Level AP)

Definition and development of key product and process measurements that can be used to validate the required level of software assurance appropriate to a given life-cycle phase

3.2.2. Other performance indicators that test for the baseline as defined in 3.1.1., by life-cycle phase (Bloom Level AP)

Definition and development of additional performance indicators that can be used to validate the required level of software assurance appropriate to a given life-cycle phase

3.2.3. Measurement processes and frameworks (Bloom Level C)

Knowledge of the range of software assurance measurement processes and frameworks and how these might be used to accomplish software assurance integration into SDLC phases

3.2.4. Business survivability and operational continuity (Bloom Level AP)

Definition and development of performance indicators that can specifically address the software/system's ability to meet business survivability and operational continuity requirements, to the extent the software affects these

3.3. Assurance Assessment Process (collect and report measures that demonstrate the baseline as defined in 3.1.1.)

3.3.1. Comparison of selected measurements to the established baseline (Bloom Level AP)

Analysis of key product and process measures and performance indicators to determine if they are within tolerance when compared to the defined baseline

3.3.2. Identification of out-of-tolerance variances (Bloom Level AP)

Identification of measures that are out of tolerance when compared to the defined baselines and ability to develop actions to reduce the variance

## 4. Assurance Management

Outcome: 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.

4.1. Making the Business Case for Assurance

4.1.1. Valuation and cost-benefit models, cost and loss avoidance, return on investment (Bloom Level AP)

Application of financially based approaches, methods, models, and tools to develop and communicate compelling cost-benefit arguments in support of deploying software assurance practices

4.1.2. Risk analysis (Bloom Level C)

Knowledge of how risk analysis can be used to develop cost-benefit arguments in support of deploying software assurance practices

4.1.3. Compliance justification (Bloom Level C)

Knowledge of how compliance with laws, regulations, standards, and policies can be used to develop cost-benefit arguments in support of deploying software assurance practices

4.1.4. Business impact/needs analysis (Bloom Level C)

Knowledge of how business impact and needs analysis can be used to develop cost-benefit arguments in support of deploying software assurance practices, specifically in support of business continuity and survivability

4.2. Managing Assurance

4.2.1. Project management across the life cycle (Bloom Level C)

Knowledge of how to lead software and system assurance efforts as an extension of normal software development (and acquisition) project management skills

4.2.2. Integration of other knowledge units (Bloom Level AN)

Identification, analysis, and selection of software assurance practices from any knowledge units that are relevant for a specific software development or acquisition project

4.3. Compliance Considerations for Assurance

4.3.1. Laws and regulations (Bloom Level C)

Knowledge of the extent to which selected laws and regulations are relevant for a specific software development or acquisition project, and how compliance might be demonstrated

4.3.2. Standards (Bloom Level C)

Knowledge of the extent to which selected standards are relevant for a specific software development or acquisition project, and how compliance might be demonstrated

4.3.3. Policies (Bloom Level C)

Knowledge of how to develop, deploy, and use organizational policies to accelerate the adoption of software assurance practices, and how compliance might be demonstrated

### 5. System Security Assurance

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

5.1. For Newly Developed and Acquired Software for Diverse Systems

5.1.1. Security and safety aspects of computer-intensive critical infrastructure (Bloom Level K)

Knowledge of safety and security risks associated with critical infrastructure systems such as found, for example, in banking and finance, energy production and distribution, telecommunications, and transportation systems

5.1.2. Potential attack methods (Bloom Level C)

Knowledge of the variety of methods by which attackers can damage software or data associated with that software by exploiting weaknesses in the system design or implementation

5.1.3. Analysis of threats to software (Bloom Level AP)

Analysis of the threats to which software is most likely to be vulnerable in specific operating environments and domains

5.1.4. Methods of defense (Bloom Level AP)

Familiarity with appropriate countermeasures such as layers, access controls, privileges, intrusion detection, encryption, and code review checklists

5.2. For Diverse Operational (Existing) Systems

5.2.1. Historic and potential operational attack methods (Bloom Level C)

Knowledge of and ability to duplicate the attacks that have been used to interfere with an application's or system's operations

5.2.2. Analysis of threats to operational environments (Bloom Level AN)

Analysis of the threats to which software is most likely to be vulnerable in specific operating environments and domains

5.2.3. Design of and plan for access control, privileges, and authentication (Bloom Level AP)

Design of and plan for access control and authentication

5.2.4. Security methods for physical and personnel environments (Bloom Level AP)

Knowledge of how physical access restrictions, guards, background checks, and personnel monitoring can address risks

5.3. Ethics and Integrity in Creation, Acquisition, and Operation of Software Systems

5.3.1. Overview of ethics, code of ethics, and legal constraints (Bloom Level C)

Knowledge of how people who are knowledgeable about attack and prevention methods are obligated to use their abilities, both legally and ethically, referencing the Software Engineering Code of Ethical and Professional Conduct [ACM 2009]

5.3.2. Computer attack case studies (Bloom Level C)

Knowledge of the legal and ethical considerations involved in analyzing a variety of historical events and investigations

### 6. System Functionality Assurance

Outcome: Graduates will have the ability to verify new and existing software system functionality for conformance to requirements and to help reveal malicious content.

6.1. Assurance Technology

6.1.1. Technology evaluation (Bloom Level AN)

Evaluation of capabilities and limitations of technical environments, languages, and tools with respect to creating assured software functionality and security

6.1.2. Technology improvement (Bloom Level AP)

Recommendation of improvements in technology as necessary within project constraints

6.2. Assured Software Development

6.2.1. Development methods (Bloom Level AP)

Rigorous methods for system requirements, specification, architecture, design, implementation, verification, and testing to develop assured software

6.2.2. Quality attributes (Bloom Level C)

Software quality attributes and how to achieve them

6.2.3. Maintenance methods (Bloom Level AP)

Assurance aspects of software maintenance and evolution

6.3. Assured Software Analytics

6.3.1. Systems analysis (Bloom Level AP)

Analysis of system architectures, networks, and databases for assurance properties

6.3.2. Structural analysis (Bloom Level AP)

Structuring the logic of existing software to improve understandability and modifiability

6.3.3. Functional analysis (Bloom Level AP)

Reverse engineering of existing software to determine functionality and security properties

6.3.4. Analysis of methods and tools (Bloom Level C)

Capabilities and limitations of methods and tools for software analysis

6.3.5. Testing for assurance (Bloom Level AN)

Evaluation of testing methods, plans, and results for assuring software

6.3.6. Assurance evidence (Bloom Level AP)

Development of auditable assurance evidence

6.4. Assurance in Acquisition

6.4.1. Assurance of acquired software (Bloom Level AP)

Assurance of software acquired through supply chains,[16] vendors, and open sources, including developing requirements and assuring delivered functionality and security

6.4.2. Assurance of software services (Bloom Level AP)

Development of service level agreements for functionality and security with service providers and for monitoring compliance

## 7. System Operational Assurance

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

7.1. Operational Procedures

7.1.1. Business objectives (Bloom Level C)

Role of business objectives and strategic planning in system assurance

7.1.2. Assurance procedures (Bloom Level AP)

Creation of security policies and procedures for system operations

7.1.3. Assurance training (Bloom Level K)

Selection of training for users and system administrative personnel in secure system operations

7.2. Operational Monitoring

7.2.1. Monitoring technology (Bloom Level C)

Capabilities and limitations of monitoring technologies, and installation and configuration or acquisition of monitors and controls for systems, services, and personnel

7.2.2. Operational evaluation (Bloom Level AP)

Evaluation of operational monitoring results with respect to system and service functionality and security

7.2.3. Operational maintenance (Bloom Level AP)

Maintenance and evolution of operational systems while preserving assured functionality and security

7.2.4. Malware analysis (Bloom Level AP)

Evaluation of malicious content and application of countermeasures

7.3. System Control

7.3.1. Responses to adverse events (Bloom Level AN)

Plan for and execution of effective responses to operational system accidents, failures, and intrusions

---

[16] For more information about software security supply chain risk, download the SEI report *Evaluating and Mitigating Software Supply Chain Security Risks* [Ellison 2010].

7.3.2. Business survivability (Bloom Level AP)

Maintenance of business survivability and continuity of operations in adverse environments (see also Knowledge Unit 3, Assurance Assessment)

Having a defined set of student prerequisites, established outcomes, a core body of knowledge, and curriculum architecture is necessary but not sufficient. Often the most challenging part of putting a new program or a new track in place is implementation. The next section provides guidelines and recommendations for faculty members to consider when considering starting an MSwA program.

# 8 Implementation Guidelines

## Issues to Address when Implementing a Graduate Software Assurance Program

There are several issues to consider when implementing any new academic program. In addition, software assurance programs have a few special challenges that need to be addressed. The main categories of issues are

- planning and launching a new program
- recruiting and preparing students
- finding and training faculty
- acquiring resources
- designing capstone courses

### Planning and Launching a New Program

A prerequisite for starting any successful program is a champion who will lead the effort. This might be a faculty member, a department head, a dean, or another member of the academic community dedicated to starting the program. In addition, it helps to have other champions from industry and government who will support the program, perhaps by voicing support to others, hiring graduates, or providing resources. If possible, it is advisable to form an industry advisory board (IAB) early on to help support and shape the program.

The academic champion needs to make a convincing case for the program by preparing a business plan, including a market study. The plan should be used to convince university colleagues and administrators that there will be sufficient interest in the program and that graduates will be successful in their career plans. Competing programs should be identified, some of which may be on the same campus.

New programs need to be sold at several levels of campus administration and even at state levels in some cases. For example, some states require extensive proposals for new academic programs, including details about courses, faculty, and dedicated resources. It is often much easier to get approval to create a new track within an existing program than it is to create a new program.

There are federal government assistance programs, such as the Federal Cyber Service Scholarship for Service program that may help [OPM 2010]. These programs provide some financial assistance to students and help justify the need for new academic programs. There are also federal agencies (for example, the National Science Foundation and the Department of Education) that provide start-up funds for innovative educational programs.

### Recruiting and Preparing Students

If you build it, they may not come. Recruitment of students needs to be a continual process, with effort expended every year to attract students to the program. A good market study that estimates the number of expected new students to the institution and current students who may enroll in the program should identify the likely areas from which to draw students. An IAB can help keep the study up to date and provide some additional help in recruiting.

Since some of the potential students are already in the workforce, it is helpful to establish relationships with the human resources (HR) department of likely employers, including those that regularly recruit students from your institution. HR departments administer benefits, such as reimbursement for tuition, and often provide information to employees about educational opportunities. It may be possible to give in-house presentations to local companies, arranged through the HR department or a member of your IAB.

Local professional organizations may provide opportunities for student recruitment. Trade organizations provide networking for local professionals and often have social events sponsored by local companies. There are often opportunities to give a short presentation or set up a booth at some of these meetings.

Most universities have professionals who help recruit students. These staff members need to be informed about any new program and the types of students who fit best. Developing brochures and a web presence helps to inform both internal staff and prospective students.

Some students may need help preparing for graduate study in software assurance. There are usually two kinds of deficiencies to be addressed: knowledge deficiencies and experience deficiencies. Knowledge deficiencies can be addressed by leveling courses, such as an overview course on software and systems engineering, a survey course in current topics in software engineering, or a survey course on security. Experience deficiencies can be partially overcome by internships in industry and assistantships within the school. Special team projects in various aspects of industrial practice can be offered for cohorts of students who lack sufficient experience (for example, a project course on the use of software tools for software development and evolution or a project course on procurement, integration, and testing of open source software packages).

### Finding and Training Faculty

There are often two sources of faculty to teach in new programs of this type: (1) faculty from related areas who have knowledge and interest in teaching software assurance and (2) experienced practitioners from industry who are interested in teaching. The former are often working in computer science academic units, but they may be found in almost any discipline that uses computing. Although they may have good teaching skills, they may need some help adjusting to the professional nature of the program. Some students will have considerable experience and expect to learn about the latest methods and tools, which is why staying current in the field is important. Consulting is one good way to stay current.

The second type of faculty candidate (from industry) may need some help making the transition to teaching. If they work part time as adjunct faculty, they will need to balance the demands of two jobs. If they become full-time faculty, there may be some discomfort in taking a salary cut. In either case it is important to ensure they appreciate the benefits of an academic position.

When PhD programs in software assurance are available, it may be possible to hire graduates of those programs.

It is prudent to ramp up faculty at a pace consistent with the growth of the program. This means that some part-time faculty will be needed early on before there is enough demand to justify

hiring full-time faculty. Adjunct faculty from industry are often used as part-time faculty, but do not forget to consider faculty from other academic units at your institution.

### Acquiring Resources

Hardware and software may be provided by local companies or members of the IAB. In addition, some vendors have academic alliance programs that provide hardware or software at deep discounts. However, there should be an annual budget allocated to acquiring and maintaining computing systems. A small program should be able to share support staff with other programs.

### Designing Capstone Courses

Capstone courses in software assurance provide their own challenges. Fortunately there are several models from which to choose. One issue to resolve early on is whether the capstone course(s) will be integrated with other courses in the curriculum. Integrated capstones provide connections to several other courses in the curriculum, offering opportunities for students to practice skills they learn in those other courses. Stand-alone capstones are easier to implement because they do not have to be synchronized with the content of other courses.

In order to provide a realistic setting for a capstone course, it is helpful to have real clients. Finding clients is another recruiting activity to plan and implement each year. Another alternative is to pursue open source projects. The community of open source developers can play the role of clients, but they will usually not have the same level of commitment as a real client.

For more information about implementation considerations, consult the GSwE2009 FAQ Discussion Forum.[17] The Implementation/Execution forum[18] specifically addresses important issues for faculty members and institutions involved in implementing and executing a graduate program in software engineering. Many of these issues will be the same for implementing an MSwA degree program.

## Ways in Which Industry Can Support Software Assurance Education

For degree programs targeted toward professionals, such as the MSwA, industry support is essential. In addition to participating in industry advisory boards, making donations, or providing discounts on equipment and software, there are a number of other ways in which industry can contribute towards advancing this new discipline. These include

- encouraging employees to work with universities as adjunct faculty or guest lecturers. This can enrich both the industry organization and the university program.
- sponsoring and speaking at faculty development workshops. It is important to provide faculty development workshops for those who wish to teach a new discipline. However, the cost of such workshops can be significant. Industry could assist with the cost, help to shape the material, and provide guest speakers for such workshops.

[17] See http://www.gswe2009.org/faq/#cat5.

[18] See http://www.gswe2009.org/faq/?tx_mmforum_pi1[action]=list_topic&tx_mmforum_pi1[fid]=8.

- providing grants to help develop new degree programs. Implementing new degree programs is very expensive, and assistance with some of the development costs could help get a new program off the ground.
- providing scholarships and summer internships to students in these programs. This is a good way to ensure that graduates can hit the ground running once they complete their degree program.
- providing support for realistic capstone projects. Industry could provide valuable support by proposing capstone problems, acting as a client, reviewing deliverables, and/or furnishing advice about project management, development methods, and technology.
- modifying and updating employee position descriptions to raise the bar. Many industry position descriptions focus on low-level skills, such as the ability to code in C or Java, and do not highlight more advanced skills needed to produce assured software, such as background in risk analysis, attack patterns, threat modeling, and secure programming and testing.
- creating an endowed chair position in software assurance. An endowed position would ensure longevity for the program.

The work described in this report can serve as a solid foundation for developing a master's degree program in software assurance. We close the report with concrete suggestions for moving forward, including ensuring that members of the target audience are aware of this project through broad-based and targeted communication, enabling MSwA2010 curriculum use and application, ensuring that the MSwA2010 curriculum is accepted as a standard model for graduate software assurance curriculum development, and making sure this content is regularly reviewed and refreshed.

# 9 Next Steps and Dissemination

This report is just the first step in the set of activities needed to support Master of Software Assurance degree programs and tracks. In this section, we describe the additional activities that are needed to support dissemination of the curriculum into the computing education community and transition into actual degree programs and tracks.

In order for the MSwA2010 curriculum to be considered successful, the curriculum model must be available, understood by the targeted academic and industrial communities, viewed as a key reference for software assurance curriculum development, and actually used to develop and modify software-assurance-focused curricula. In the following subsections, we discuss various approaches for making this model available, understood, and used. The discussion might be viewed as a plan for what follows the completion of this report; that is—*What are the next steps?*

### The Computing Education Community Knows About MSwA2010

In 2010, one of the primary activities of the MSwA2010 project will be disseminating MSwA2010 information. The following are some of the planned activities:

- Conduct tutorials and workshops and present information papers about MSwA2010 at national and international computing and engineering education conferences and workshops. Possible conferences include the Conference on Software Engineering Education and Training, the Association for Computing Machinery (ACM) Technical Symposium on Computer Science Education, The Colloquium for Information Systems Security Education, the Frontiers in Education conference, and the European Association for Education in Electrical and Information Engineering conference.
- Publish technical papers and articles about MSwA2010 in the journals, magazines, and newsletters of professional organizations, government agencies, and research institutes. Examples of such organizations include ACM, IEEE Computer Society (IEEE-CS), American Society for Engineering Education (ASEE), SEI, DHS, and DoD.
- Post this report and general information about MSwA2010 on various websites such as at the SEI and DHS. Communicate through various listservers, discussion boards, and webinars to describe MSwA2010 features and provide forums for discussion.

### MSwA2010 Is Used

A critical element of the MSwA2010 project is to enable educational institutions to use MSwA2010. The previous section discusses a number of ideas that would advance this goal. The following are some additional activities that would help a start-up program:

- identify programs and individuals likely to use MSwA2010 elements
- solicit current MSE programs to conduct a trial review of the MSwA2010 and provide information and opinion about how MSwA2010 elements could be incorporated into their curricula

- conduct a multi-day workshop for faculty interested in using MSwA2010 to implement a new MSwA program or track, or who would like to integrate elements of MSwA2010 into an existing program
- provide mentors and advisors to programs to help them in the use and application of MSwA2010

### MSwA2010 Is Accepted as a Standard

In order for MSwA2010 to have significant influence on the state of software assurance education, it is important for it to be accepted as the standard model for graduate efforts in professional software assurance education. Activities that support such acceptance include general knowledge and understanding of MSwA2010, use and application of its features, and formal recognition by professional computing organizations (for example, ACM and IEEE-CS). We will seek such formal recognition in the coming year.

### MSwA2010 Stays Current

Software assurance methods and technology are dynamic and evolving. In order for a professional software assurance curriculum to stay current, it is important for it to be regularly assessed and updated as the discipline changes and advances. For MSwA2010 to remain viable, it must be reviewed and updated at regular intervals. Such maintenance will require long-term stewardship by an appropriate organization. Communication and agreement with such an organization is another activity for the coming year.

# Appendix A: Bloom's Taxonomy and the GSwE2009

Bloom's Taxonomy is a classification system devised in 1956 by group of educators led by Benjamin Bloom [Bloom 1956]. The taxonomy can be used by educators to set the level of educational and learning objectives required for students engaged in an education unit, course, or program. Bloom's Taxonomy divides educational objectives into three domains: affective, psychomotor, and cognitive. In this report, the focus is on the cognitive domain, which is concerned with what we know and how we know it [Huitt 2006]. Conventional education systems tend to stress outcomes in the cognitive domain, particularly the lower-level objectives.

Bloom's taxonomy is hierarchical; that is, learning at a higher level is dependent on attaining prerequisite knowledge and skills at the lower levels. Table 1 provides a description of the Bloom's Levels for the Cognitive Domain.

Note: This appendix was adapted from an appendix in the GSwE2009 [iSSec 2009].

*Table 1: Bloom's Taxonomy*

| Level | Competency | Objective Descriptors |
|---|---|---|
| Knowledge (K) | (Lowest level) Remembering previously learned material. Test observation and recall of information, i.e., "bring to mind the appropriate information" (e.g., dates, events, places, knowledge of major ideas, mastery of subject matter). | list, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name (who, when, where, etc.) |
| Comprehension (C) | Understanding information and ability to grasp meaning of material presented. For example, translate knowledge into new context, interpret facts, compare, contrast, order, group, infer causes, predict consequences, etc. | summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend |
| Application (AP) | Ability to use learned material in new and concrete situations. For example, use information, methods, concepts, and theories to solve problems requiring the skills or knowledge presented. | apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover |
| Analysis (AN) | Ability to decompose learned material into constituent parts in order to understand structure of the whole. This includes seeing patterns, organization of parts, recognition of hidden meanings, and identification of parts. | analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer |
| Synthesis (S) | Ability to put parts together to form a new whole. This involves using existing ideas to create new ones, generalizing from facts, relating knowledge from several areas, and predicting and drawing conclusions. It may also involve adapting general solution principles to the embodiment of a specific problem. | combine, integrate, modify, rearrange, substitute, plan, create, design, invent, what if?, compose, formulate, prepare, generalize, rewrite |
| Evaluation (E) | (Highest level) Ability to pass judgment on value of material within a given context or purpose. This involves making comparisons and discriminating between ideas, assessing value of theories, making choices based on reasoned arguments, verifying value of evidence, and recognizing subjectivity. | assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize |

# Appendix B: Coverage of the Practices by the Core Body of Knowledge

This appendix resulted from process steps 2, 3, 4, and 7, as described in Section 2. It includes a series of ten tables as follows:

- Summary Table: Knowledge Unit Coverage of SDLC Practices. This table contains a summary of all tables to follow, reflecting the table name (most often an SDLC phase), the categories within that table's topic, and a cross reference to the applicable BoK knowledge units (KU) by category.
- Table 1: Software Security Practices That Span the SDLC. This table, and all subsequent tables, reflect the references from which the practices were drawn (with citations), a brief description (with citations), the categories within that table's topic, and a cross reference to the applicable BoK knowledge units by category.
- Table 2: Requirements Engineering Practices
- Table 3: Architecture and Design Practices
- Table 4: Coding Practices
- Table 5: Testing Practices
- Table 6: Analysis of Software and Services in Static and Operational Contexts. In this table, references are replaced by subject matter, indicating the object of the analysis.
- Table 7: Assembly, Evolution, and Deployment
- Table 8: Risk Mitigation Strategies for System Complexity and Scale
- Table 9: Governance and Management Practices

As described in Section 2, the purpose of the practices-to-knowledge-unit gap analysis (reflected in the column titled "Applicable Knowledge Units") was to ensure that all practice categories were covered by at least one knowledge unit or that the MSwA2010 team made a conscious decision to exclude some practice topic we thought was out of scope, for example, privacy. This exercise was neither exhaustive nor rigorous, and the result should not be construed as a complete traceability.

*Knowledge Unit Coverage of SDLC Practices*

| Life-Cycle Phase/Topic | Categories | Applicable Knowledge Units |
|---|---|---|
| Table 1: Software Security Practices that Span the SDLC | Fundamentals | Fundamentals [6.1.1,6.2.1,6.3.1, 6.4.1, 6.4.2, 7.1.1] |
| | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.2.2, 5.3.2] |
| | Evidence | Evidence [3.1.2, 6.3.6] |
| Table 2: Requirements Engineering Practices | Fundamentals | Fundamentals [5.2.2, 7.1.1] |
| | Process | Process [1.1.1, 1.1.3, 1.2.2, 3.1.2, 6.2.1, 6.2.3, 7.1.2] |
| | Elicitation | Elicitation [2.2.2, 2.3.2, 6.2.1, 6.2.2] |
| | Analysis | Analysis [2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.3.1, 2.3.2, 3.1.1, 3.1.2, 3.3.1, 3.3.2, 5.1.3, 5.1.4, 5.2.2, 6.2.1, |

| Life-Cycle Phase/Topic | Categories | Applicable Knowledge Units |
|---|---|---|
| | | 7.1.1, 7.2.1, 7.2.2, 7.3.1, 7.3.2] |
| | Specification | Specification [5.1.4, 6.2.1] |
| | Validation | Validation [3.1.1, 3.1.2, 3.3.1, 3.3.2, 6.2.1, 6.2.2, 6.4.1, 6.4.2] |
| Table 3: Architecture and Design Practices | Design concepts | Design concepts [5.1.4, 6.2.1, 6.3.1] |
| | Architecture | Architecture [5.1.4, 5.2.3, 6.2.1, 6.3.1] |
| | Module/component design | Module/component design [5.1.4, 6.2.1] |
| | Detailed design | Detailed design [5.1.4, 6.2.1] |
| | Design review/assessment | Design review/assessment [6.2.1] |
| Table 4: Coding Practices | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| | Metric analysis | Metric analysis [3.2.1, 3.2.2, 3.2.3] |
| | Environment-specific risks | Environment-specific risks [2.1.1, 2.3.1, 5.1.1, 5.2.1, 5.2.2, 5.3.2, 7.1.1] |
| Table 5: Testing Practices | Security testing (penetration, cases based on requirements)<br><br>Security unit testing (white box)<br><br>Security integration testing<br><br>Security functional testing<br><br>Code coverage analysis<br><br>Black box security tools<br><br>Fuzz testing | Security testing, all categories [3.1.1, 3.1.2, 3.3.1, 3.3.2, 4.1.4, 5.1.2, 5.2.1, 5.3.2, 6.2.1, 6.2.3, 6.3.5] |
| Table 6: Analysis of Software and Services in Static and Operational Contexts | Technologies and methods | |
| | Threat, vulnerability, and security assessment | Threat, vulnerability, and security assessment [3.1.2, 3.3.1, 3.3.2, 5.1.2, 5.1.3] |
| | Requirements specification, architecture, design, and code analysis | Requirements specification, architecture, design, and code analysis [5.1.3, 6.2.1] |
| | Flow and service analysis | Flow and service analysis [5.2.2, 6.2.1, 6.3.2, 6.3.3, 6.3.4, 7.2.1, 7.2.2., 7.2.3, 7.2.4] |
| | Reverse engineering | Reverse engineering [6.3.1, 6.3.2, 6.3.3, 6.3.4] |
| | Verification and testing of security and functionality | Verification and testing of security and functionality [6.2.1, 6.3.5, 6.4.1, 6.4.2] |
| | Application of standards and practices | Application of standards and practices [1.2.1] |
| | Applied to software, services, data, networks, humans, and operations | |
| | For system types: systems, systems-of-systems, distributed systems, SOA and cloud systems, | Systems, systems-of-systems, distributed systems, SOA and cloud systems, infrastructure systems, |

| Life-Cycle Phase/Topic | Categories | Applicable Knowledge Units |
|---|---|---|
| | infrastructure systems, embedded systems | embedded systems [5.1.1, 6.4.1, 6.4.2] |
| Table 7: Assembly, Evolution, and Deployment | Incremental development and evolution | Incremental development and evolution [1.1.1, 1.1.2, 1.1.3, 6.2.1] |
| | Systems integration | System integration [6.2.1] |
| | User training | User training [7.1.3] |
| | Maintenance and patching | Maintenance and patching [6.2.3] |
| | System monitoring and management | System monitoring and management [5.2.1, 5.2.2, 7.2.1, 7.2.2, 7.2.3, 7.2.4, 7.3.1, 7.3.2 ] |
| Table 8: Risk Mitigation Strategies for System Complexity and Scale | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | Integration | Integration [6.2.1] |
| | User training | User training [7.1.3] |
| | Maintenance and patching | Maintenance and patching [7.2.3] |
| | System monitoring and management | System monitoring and management [7.2.1, 7.2.2] |
| Table 9: Governance and Management | Business case | Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4, 7.1.1, 7.1.2] |
| | Risk management | Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1] |
| | Awareness | Awareness [part of all KUs; 5.1.2] Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4] (Awareness for business leaders) |
| | Training, education, certification | Training, education, certification [1.2.1, 7.1.3] |
| | Project management (process management) | Project management [1.1.1, 1.2.2, 4.2.1, 4.2.2, 7.1.1, 7.1.2] |
| | SwA practices integrated with SDLC | SwA practices/process [1.1.1, 1.1.2, 1.1.3, 1.2.1, 1.2.2, 6.2.1, 7.1.2] |
| | Transition and adoption | Adoption [1.1.1, 1.1.2, 1.2.1, 1.2.2, 7.1.1] |
| | Measurement | Measurement [1.2.2, 3.2.1, 3.2.2, 3.2.3, 6.3.6] |
| | Ethics | Ethics [5.3.1] |
| | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3, 6.3.6] |
| | Evaluation (systems, software, people) | Evaluation [3.1.2, 3.3.1, 3.3.2, 5.2.4, 6.1.1, 6.3.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5, 6.3.6] |
| | Acquisition | Acquisition [1.1.3, 1.2.1, 1.2.2, 6.4.1, 6.4.2] |

*Table 1: Software Security Practices That Span the SDLC*

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Fundamentals | Fundamentals [6.1.1, 6.2.1, 6.3.1, 6.4.1, 6.4.2, 7.1.1] |
| | | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.2.2, 5.3.2] |
| | | Evidence | Evidence [3.1.2, 6.3.6] |
| Properties of secure software [1] [6] | Core and influential properties of software that enable the understanding and description of its security characteristics [1] [6] | Fundamentals | Fundamentals [6.2.2] |
| SAFECode [23] | Integrity principles: least privilege access, separation of duties, chain of custody and supply chain integrity; persistent protection; compliance management [23] | Fundamentals | Fundamentals [6.2.2, 6.4.1, 6.4.2] Compliance [4.3.1, 4.3.2, 4.3.3] |
| Guiding security and privacy principles: Microsoft Software Development Lifecycle [4] | SDL reflects the SD3+C principles: secure by design (four principles), secure by default (five principles), secure in deployment (three principles), and communications (two principles). PD3+C includes privacy by design (four principles), privacy by default (one principle), privacy in deployment (one principle), and communications (three principles). [4] | Fundamentals (Privacy OK at this level but likely no further.) | Fundamentals [6.2.2] |
| Fundamental concepts and principles: SwACBK [14] [20] [27] | Basic concepts (dependability, security, assurance, etc.); 14 system security principles (least privilege, etc); safety; secure software engineering; security properties (CAI, accountability) [14] [20] [27] | Fundamentals (Safety OK at this level but no further.) | Fundamentals [6.2.2] |
| Other general topics: James Madison University [6] | Network security, controls, information assurance<br>Cryptography: basics and introduction to deciding on techniques to use<br>Fundamentals of computer security: access control, confidentiality, integrity, etc. [6] | Fundamentals | Fundamentals [5.2.2, 5.1.4, 5.3.2] |
| IEEE article: Yasar et al [20] | Goals of software security: prevention, traceability and auditing, monitoring, privacy and confidentiality, multi-level security, anonymity, authentication, integrity [20] | Fundamentals | Fundamentals [6.2.2] |
| *Computing Curriculum Series* [8] | The theory and application of access control to computer systems and the information contained in them [8] | Fundamentals | Fundamentals [5.2.3] |
| *Computing Curriculum Series* [10] | Concepts of information assurance, including data persistence, integrity, etc. [10] | Fundamentals | Fundamentals [6.2.2] |
| *Computing Curriculum Series* [11] | Cryptography, forensics, and biometrics [11] | Fundamentals | Fundamentals [5.1.4] |
| *Computing Curriculum Series* [10] | Network security for net-centric computing environments [10] | Fundamentals—special topics | Fundamentals [5.2.1] |

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| *Computing Curriculum Series* [9] | Data security and protection [9] | Fundamentals—special topics | Fundamentals [5.2.3] |
| *Computing Curriculum Series* [9, 10] | Security and protection considerations associated with operating systems [9, 10] | Fundamentals—special topics | Fundamentals [5.1.4] |
| CMU—Heinz [21] | Computer system vulnerabilities; effective cryptographic techniques and protocols; access control policies and mechanisms; and implications of security technology in the realm of risk management [21]<br>Design and implementation of computer security policies and standards, disaster recovery plans, system security architectures and physical security controls, legal aspects of computer system auditing in a secure environment, management of a site's computer security on a daily basis [21] | A mixture, fundamentals | Fundamentals [5.1.2, 5.1.4, 5.2.1, 5.2.3, 5.3.2]<br>Risk management [2.3.1, 2.3.2, 2.3.3, 2.3.4, 4.1.2, 7.3.1, 7.3.2]<br>Design and implementation [6.2.1, 6.4.1, 6.4.2, 7.1.2] |
| CMU—Information Networking Institute (INI) [21] | Introduction to techniques for defending against hostile adversaries in modern computer systems and networks: operating system security, network security, firewalls, denial-of-service attacks, user authentication, network server and mobile security [21] | A mixture, fundamentals | Fundamentals [5.1.4, 5.2.3, 5.2.4] |
| | | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.3.2] |
| ISC$^2$ Certified Secure Software Lifecycle Professional (CSSLP) [22] | Confidentiality, availability, integrity, (CAI); authentication, authorization, auditing; security design principles [11]; risk management; regulations; privacy and compliance; software architecture; software development methodologies; legal; standards; security models (e.g., Bell-Lapadula); trusted computing; acquisition [22] | Fundamentals | Fundamentals [5.1.4, 5.2.2, 5.2.3, 5.3.1]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 4.1.2, 7.3.1, 7.3.2]<br>Compliance [4.3.1, 4.3.2 4.3.3] |
| Security engineering: Anderson [16] | Usability and psychology: psychological attacks, perceptual bias, mental process, social psychology (Sections 2.2, 2.3) [16] | Think like an attacker | Usability and psychology [7.1.3]<br>Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.3.2] |
| | Economics: information economics, game theory, security and dependability (Sections 7.3, 7.4, 7.5, 2.3) [16] | Fundamentals—special topics | Economics [7.1.1] |
| Dangers and damage: SwACBK [14] | Attackers (types, motivations); attack methods (malicious code, hidden software, social engineering, physical); non-malicious dangers; attacks across the life cycle; known vulnerabilities/exploits [14] | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.3.2] |
| Attack patterns [1] | Formalized capture of common methods of attacking software to serve as guides for improving software attack resistance and resilience [1] | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.3.2] |
| Building Security In Maturity Model (BSIMM) domain, | Attack models: think like an attacker; threat modeling; abuse case development/refinement; data | Think like an attacker | Think like an attacker [5.1.1, 5.1.2, 5.2.1, 5.3.2] |

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| intelligence [2] | classification; technology-specific attack patterns | | Threat modeling [3.1.2, 3.3.1, 3.3.2] |
| OWASP Software Assurance Maturity Model (SAMM), construction [3] | Threat assessment: identify high-level threats; build threat models, attacker profiles, abuse case models; evaluate third-party software risk; add compensating controls [3] | Think like an attacker | Think like an attacker [5.1.3, 5.2.2]<br>Threat assessment/risk evaluation [3.1.2, 3.3.1, 3.3.2]<br>Evaluate third party software risk [2.2.2, 2.2.4] |
| Assurance cases [1] [6] | Structured mechanism for capturing, communicating, and validating desired or attained levels of software security assurance in terms of the properties of secure software [1] [6] | Evidence | Evidence [3.1.2, 6.3.6] |
| Assurance cases: JMU [6] | Assurance cases: top-level claim such as a safety- or security-related claim, the arguments for this claim, and the evidence that supports these arguments [6] | Evidence | Evidence [3.1.2, 6.3.6] |

*Table 2: Requirements Engineering Practices*

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Fundamentals | Fundamentals [5.2.2, 7.1.1] |
| | | Process | Process [1.1.1, 1.1.3, 1.2.2, 3.1.2, 6.2.1, 6.2.3, 7.1.2] |
| | | Elicitation | Elicitation [2.2.2, 2.3.2, 6.2.1, 6.2.2] |
| | | Analysis | Analysis [2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.3.1, 2.3.2, 3.1.1, 3.1.2, 3.3.1, 3.3.2, 5.1.3, 5.1.4, 5.2.2, 6.2.1, 7.1.1, 7.2.1, 7.2.2, 7.3.1, 7.3.2] |
| | | Specification | Specification [5.1.4, 6.2.1] |
| | | Validation | Validation [3.1.1, 3.1.2, 3.3.1, 3.3.2, 6.2.1, 6.2.2, 6.4.1, 6.4.2] |
| | | Fundamentals | Fundamentals [5.2.2, 7.1.1]<br>Threat objectives [3.1.1] |
| Requirements: Bishop [07] | Concept of operations: business and organization issues, high-level requirements, threat and security objectives [Section 18.2.1.1, 19.1.1] [07] | Fundamentals | Fundamentals [5.2.2, 7.1.1]<br>Threat objectives [3.1.1] |
| | Analysis: security requirements, feasibility study, prototyping, vulnerability analysis [Sections 18.1.2, 18.2.2.1, 18.2.3.1, 19.2.3.3, 23.1, 23.2] [07] | Analysis | Vulnerability analysis [3.1.2, 3.3.1, 3.3.2] |
| | Specification: technical, functional, formal, traceability [Sections 18.2.2.1, 19.2.2.2, 19.2.4.1, 20.2] [07] | Specification | Specification [6.2.1] |
| Security engineering, requirements engineering: Anderson [16] | Security requirements engineering: managing requirements, requirements evolution (Section 25.4) [16] | Process | Process [1.1.1, 1.2.2, 6.2.3] |
| Standard security requirements engineering process [1] | Establish a defined process for identifying and documenting security requirements, such as Security Quality Requirements Engineering (SQUARE) | Process | Process [1.2.2, 6.2.1] |
| Secure software requirements: SwACBK [14] | Identify needs (stakeholder, asset, threat, usability, etc.); requirements analyses (risk, feasibility, tradeoffs); specification; validation; assurance case [14] | Process | Assurance case [3.1.2]<br>Risk [2.2.1, 2.2.2, 2.2.3, 2.2.4, 7.3.1, 7.3.2] |
| Security risk assessment [1] [6] | Perform a risk assessment aimed at security exposures, either as part of a project risk assessment or as a stand-alone activity [1] [6] | Process, analysis | Process, analysis [5.1.3, 5.2.2]<br>Risk assessment [3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.3.1, 7.3.2]<br>Risk assessment [2.3.1, 2.3.2] |

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Assurance for CMMI, engineering [18] | Understand the operating environment and define operating constraints for assurance within environments of system deployment; develop customer assurance requirements; define product assurance requirements; identify operational concepts and scenarios for assurance; analyze assurance requirements; balance assurance needs against cost benefits [18] | Process, analysis | Process, analysis [5.1.3, 5.2.2, 7.1.1, 7.2.1, 7.2.2] |
| OWASP SAMM, construction [3] | Security requirements: specify security requirements based on business functionality and known risks; reflect requirements in supplier agreements; expand audit program [3] | Process + others | Known risk [2.3.1, 2.3.2, 7.1.1] |
| JMU [6] | Identify stakeholder security-related needs; asset protection needs; threat analysis; interface and environment requirements; usability needs; reliability needs; availability, tolerance and survivability needs; sustainability (maintainability) needs; deception needs; validity, verifiability, and evaluation needs; certification needs; system accreditation and auditing needs; analysis of conflicts among security needs [6]<br><br>Specify software-related security policy and security functionality requirements [6] | Process, analysis, specification, validation | Tolerance [3.1.1]<br><br>Threat analysis/validation [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br><br>Survivability [3.2.4, 73.1, 7.3.2] |
| Phase 1 requirements: Microsoft SDL [4] | Define quality gates/bug bar (minimum level of quality); analyze security and privacy risk; identify security requirements; identify privacy requirements [4] | Process, elicitation, specification, analysis | Risk analysis [3.1.2, 3.3.1, 3.3.2]<br>Security and privacy risk [2.2.2, 2.3.2] |
| Security requirements elicitation [1] [6] | Conduct a security requirements elicitation activity to identify potential security requirements [1] [6] | Elicitation | Requirements [6.2.1] |
| BSIMM domain, intelligence [2] | Standards and requirements: elicit security requirements [2] | Elicitation | Elicitation [6.2.1] |
| | Recommend COTS; develop standards for security controls (authentication, input validation, etc.); develop standards for technologies in use; create standards review board [2] | Analysis, specification | Analysis, specification [5.1.4]<br>Standards [1.2.1, 1.2.2, 6.1.1, 6.1.2] |
| SAFECode [23] | Identify requirements from use cases, customer inputs, company policy, best practices, and security improvement goals (provides a list of 12 areas that requirements should cover) [23] | Elicitation | Elicitation [6.2.1] |
| | Prioritize requirements based on threat and risk levels [23] | Analysis | Threat and risk levels [3.1.1]<br>Risk levels [2.1.2, 2.2.2, 2.3.2, 7.1.1] |
| ISC$^2$ CSSLP [22] | Policy decomposition: CAI requirements; authentication, authorization, auditing requirements; | Elicitation | Requirements [6.2.1] |

| References | Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | internal and external requirements [22] | | |
| | Identification and gathering: data classification; use cases; abuse cases (inside and outside adversaries) [22] | Analysis | Analysis [5.1.2, 5.1.3, 5.1.4] |
| *Computing Curriculum Series* [10] | Specification of human-computer interactions to protect against insecure use of software application [10] | Specification | Specification [6.2.1, 6.2.2] |
| Threat identification [1] [6] | Use techniques such as misuse/abuse cases, threat modeling, attack patterns, or attack trees to identify security threats [1] [6] | Analysis | Analysis [5.1.3, 5.2.2, 5.3.2]<br>Threat modeling [3.1.2, 3.3.1, 3.3.2] |
| Security requirements categorization and prioritization [1] [6] | Categorize and prioritize security requirements to separate true requirements from architectural recommendations and to optimize cost-benefit considerations [1] [6] | Analysis | Requirements [6.2.1] |
| *Industrial Network Security* [19] | Identify cybersecurity issues as related to safety of industrial networks [19] | Analysis | Analysis [5.1.1] |
| Security requirements inspection [1] [6] | Inspect security requirements in conjunction with other requirements to ensure they are correct and complete [1] [6] | Validation | Validation [3.1.1, 3.1.2, 3.3.1, 3.3.2, 6.2.1] |

*Table 3: Architecture and Design Practices*

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Design concepts | Design concepts [5.1.4, 6.2.1, 6.3.1] |
| | | Architecture | Architecture [5.1.4, 5.2.3, 6.2.1, 6.3.1] |
| | | Module/component design | Module/component design [5.1.4, 6.2.1] |
| | | Detailed design | Detailed design [5.1.4, 6.2.1] |
| | | Design review/assessment | Design review/assessment [6.2.1] |
| JMU [6] | Principles and guidelines for designing secure software; principles-based review; access control issues; cross-domain control; identity management; proper use of encryption and encryption protocols; design patterns for secure software; deception and diversion; forensic support; assurance cases for design; secure design processes and methods; design reviews for security; trust management: organizations and software entities and the implications of their trustworthiness; tolerance and defense in depth: includes redundancy, diversity, separation, and many design concerns, such as damage containment [6] | A mixture | A mixture [5.1.4, 5.2.3, 5.2.4, 6.2.1, 6.3.1] |
| | | Overarching SwA strategies, principles | Assurance cases [3.1.2]<br>Process [1.2.2] |
| SAFECode [23] | Threat analysis (aka threat modeling, risks analysis); use/misuse cases to drive understanding of how attackers might attack a system [23] | Design concepts | Design concepts [5.1.3, 5.2.3, 5.3.2]<br>Threat/risk analysis [3.1.2, 3.3.1, 3.3.2]<br>Risk analysis [2.3.1, 2.3.2] |
| | Select standard proven security toolkits such as cryptographic and protocol libraries [23] | Module/component design | Module/component design [5.1.4, 6.2.1] |
| CMU—INI [21] | Choices of programming languages, operating systems, databases and distributed object platforms, common software vulnerabilities, auditing software, proving properties of software, watermarking, code obfuscation, tamper-resistant software, open and closed source development [21] | Design concepts, detailed design. Some of this may also fit in coding. | SwA environment [5.1.4, 6.1.1, 6.1.2] |
| Security principles [1] [6] | High-level perspectives/practices to provide prescriptive guidance for architecture and design [1] [6] | Design concepts | Design concepts [5.1.4]<br>Prescriptive guidance [1.2.2] |
| | | Architecture | Architecture [6.2.1, 6.3.1] |
| Security guidelines [1] | Technology-specific prescriptive guidance founded on demonstrated experience to guide integrating | Design concepts | Technology [5.1.4, 6.1.1, 6.1.2] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | security concerns into architecture and design [1] | Architecture | Architecture [6.2.1, 6.3.1] |
| Secure software design: SwACBK [14] | Design objectives; design principles and guidelines; design assumptions, decisions, and rationales; secure architecture; security functionality; encryption; deception and diversion; assurance cases; design reviews [14] | Design concepts | Design concepts [5.1.4, 6.2.1, 6.3.1] Assurance cases [3.1.2] |
| Design: Bishop [07] | Design principles: least privilege, fail-safe defaults, economy of mechanism, complete mediation, open design, separation of privilege, least common mechanism, psychological acceptability, modularity, layering [Sections 13.2, 19.2.1] [07] specification [Sections 18.2.2.2, 19.2.2.1, 19.2.2.2] [07] | Design concepts | Design concepts [5.1.4] |
| | Architecture: component configuration, data descriptions, interface description, security function | Architecture | Architecture [5.1.4, 6.2.1, 6.3.1] |
| | Component design: interface specification, exception specification [Sections 19.2.2.4] [07] | Module/component design | Module/component design [5.1.4, 6.2.1] |
| Attack patterns [1] | Formalized capture of common methods of attacking software to serve as guides for improving the attack resistance and resilience of the software architecture [1] | Design concepts | Design concepts [5.1.2, 5.2.1, 5.3.2] |
| Security engineering, architecture and design practices: Anderson [16] | Security protocols: authentication, challenge and response, reflection attacks, message manipulations, protocol attacks, managing encryption keys (Chapter 3) [16] | Design concepts | Design concepts [5.1.4, 5.2.3, 6.2.1] |
| | Passwords: password difficulties, design errors, social-engineering attacks, phishing countermeasures, interface design, password cracking (Sections 2.4, 2.5.3, 2.5.4) [16] | Detailed design | Detailed design [5.1.4] |
| | OS access controls: groups and roles, access control lists, middleware (Sections 4.2.1 – 4.2.8) [16] | Architecture | Architecture [5.1.4] |
| | Cryptography: block ciphers, one-way functions, asymmetric ciphers, random oracle model, symmetric crypto primitives, modes of operations, asymmetric crypto primitives (Chapter 5) [16] | Detailed design | Detailed design [5.1.4] |
| | Distributive systems: concurrency issues, fault tolerance and failure recovery, naming issues (Sections 6.2, 6.3, 6.4) [16] | Module/component design | Module/component design [5.1.4, 6.2.1] |
| | Multi-level security: security policy models, information flow control, multi-level integrity models, composability, closed security environments, | Architecture | Architecture [5.1.4, 6.2.1] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | polyinstantiation (Chapter 8) [16] | | |
| | Multilateral security: compartmentalization, the Chinese Wall, the BMA model, inference control (Chapter 9) [16] | Architecture | Architecture [5.1.4, 6.2.1] |
| | Network security: protocol vulnerabilities, Trojans, viruses, worms, rootkits, firewalls, span filters, censorware, encryption (Sections 21.3, 21.4) [16] | A mixture (capture example methods of attack) | A mixture (capture example methods of attack) [5.2.3] |
| | Methodology: top-down design, iterative design (Section 25.3) [16] | Design concepts | Design concepts [5.1.4, 6.2.1]<br>Methodology [1.1.1, 6.2.1] |
| BSIMM domain, intelligence [2] | Security features and design: create usable security patterns for major security controls; build middleware frameworks for controls; create other security guidance [2] | Design concepts, architecture | Design concepts, architecture [5.1.4, 6.2.1] |
| Assurance for CMMI, engineering [18] | Develop alternative solutions and selection criteria for assurance; architect for assurance; design for assurance [18] | Architecture | Architecture [5.1.4, 6.2.1, 6.3.1] |
| Secure design patterns at the architectural level [15] | These patterns focus on high-level partitioning of responsibilities among system components and the external interaction among those components [15] | Architecture | Architecture [5.1.4, 6.2.1] |
| OWASP SAMM, construction [3] | Secure architecture: apply security principles to design; secure design patterns; security services; formal reference architecture; validate use of these [3] | Architecture | Architecture [5.1.4, 6.2.1] |
| *Computing Curriculum Series* [9] | Data security and protection [9] | Architecture | Architecture [5.1.4, 6.2.1] |
| *SwA Pocket Guide:* "Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses," Table 1 [17] | Race condition; client-side enforcement of server-side security [17] | Architecture | Architecture [5.1.4]<br>Detailed design [5.1.4] |
| *Computer Curriculum Series* [10] | Security modeling of operating system properties [10] | Architecture | Architecture [5.1.4, 6.3.1] |
| *Computing Curriculum Series* [12] | Security aspects of database design and database administration [12] | Architecture | Architecture [5.1.4, 6.3.1] |
| | | Module/component design | Module/component design [5.1.4] |
| *Industrial Network Security* [19] | Countermeasures via design/planning, technology, and people/policies/assurance [19] | Architecture | Architecture [5.1.1] |
| | | Module/component design | Module/component design [5.1.4, 6.2.1] |
| | | Assessment | Assessment [6.2.2] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Phase 2 design: Microsoft SDL [4] | Security design review; use design best practices (many details here); attack surface measurement; enable least privilege; secure default installation; defense-in-depth; security and privacy risk analysis; threat modeling [4] | A mixture | Risk analysis/threat modeling [3.1.2, 3.3.1, 3.3.2]<br>Security and privacy risk analysis [2.3.1, 2.3.2, 2.3.3, 2.3.4] |
| Secure design patterns at the design level [15] | These patterns define internal design and implementation of the parts of system components that provide security capabilities. [15] | Module/component design | Module/component design [5.1.4, 6.2.1] |
| *SwA Pocket Guide* [17] | Failure to preserve SQL query structure (SQL injection); failure to preserve web page structure (cross-site scripting); failure to preserve OS command structure (OS command injection); race condition; external control of critical state data; external control of file name or path; untrusted search path; hard-coded password [17] | Module design | Module design [5.1.4] |
| Construction: Bishop [7] | Detailed design: cryptography, key management, cipher techniques, authentication, naming and certificates, web identities, access control mechanisms, malicious logic, information flow (Chapters 9, 10, 11, 12, 15, 16, 22; Sections 14.5, 14.6) [7] | Detailed design | Detailed design [5.1.4] |
| CMU—INI [21] | Detailed study of cryptographic mechanisms: symmetric encryption, public key encryption, digital signatures, message authentication codes, crypto protocols, cryptanalysis, and further detailed crypto techniques [21] | Detailed design/assessment | Detailed design [5.1.4] |
| Architectural risk analysis (ARA) [1] | Perform a detailed risk assessment of the software architecture and design and its ability to securely support the requirements of the software [1] | Design review/assessment | Risk assessment [3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1]<br>Risk assessment [2.2.1, 2.2.2, 2.3.1, 2.3.2] |
| BSIMM domain, SSDL touchpoints [2] | Architectural analysis: capture software architecture; apply lists of risks and threats; adopt a review process (STRIDE[19], Cigital's Architectural Risk Analysis [ARA]); assessment and remediation plan [2] | Design review/assessment | Design review/assessment [1.1.2, 1.2.2, 5.1.4]<br>Architectural analysis [6.3.1]<br>Risk management [2.2.3, 2.2.4, 2.3.3, 2.3.4] |
| OWASP SAMM, verification [3] | Design review: software attack surface; design meets security requirements; design review service for project teams; release gates [3] | Design review/assessment | Design review [6.2.1] |
| *Computing* | Software quality assurance including | Design | Quality assurance |

[19] STRIDE stands for Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, Elevation of privilege. STRIDE is a threat model developed by Microsoft. See http://msdn.microsoft.com/en-us/library/ee823878%28CS.20%29.aspx.

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| *Curriculum Series* [9] | inspection of designs [9] | review/assessment | [6.2.2] |
| *SwA Pocket Guide* [17] | Improper access control (authorization); execution with unnecessary privileges [17] | Design review/assessment | Design review/assessment [5.3.2, 6.2.1] |
| ISC$^2$ CSSLP [22] | Design processes: attack surface evaluation; threat modeling, control identification and prioritization; documentation [22] | Design concepts | Design concepts [5.1.4, 6.2.1]<br>Design processes [1.2.2]<br>Threat modeling [3.1.2, 3.3.1, 3.3.2] |
| | Design considerations: CAI; authentication, authorization, and auditing; security design principles; interconnectivity; security management interfaces; identity management [22] | Design concepts | Design concepts [5.1.4, 6.2.1] |
| | Architecture: distributed computing; service-oriented architecture; rich internet applications; pervasive computing; integration with existing architectures; software as a service [22] | Architecture | Architecture [5.1.4, 6.2.1, 6.3.1] |
| | Technologies: authentication and identity management; credential management; flow control; audit; data protection; computing environment; digital rights management; integrity (code signing) [22] | Module/component design | Module/component design [5.1.4, 6.2.1] |
| | Design and architectural technical review [22] | Detailed design | Detailed design [5.1.4] |
| | | Design review/assessment | Design review/ assessment/inspection [6.2.1] |

*Table 4: Coding Practices*

| References | Brief Description | Coding Categories | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| | | Metrics analysis | Metrics analysis [3.2.1, 3.2.2, 3.2.3] |
| | | Environment-specific risks | Environment-specific risks [2.1.1, 2.3.1, 5.1.1, 5.2.1, 5.2.2, 5.3.2, 7.1.1] |
| | | Proof-carrying code | Proof-carrying code [5.1.4, 6.2.1] |
| Assurance for CMMI, engineering [18] | Implement assurance design for product components; identify deviations from assurance coding standards [18] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| Secure design patterns at the implementation level [15] | These patterns focus on specific low-level methods or functions that implement security features. [15] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| Security engineering, coding: Anderson [16] | API security: cryptographic attacks, protocol attacks, concurrency attacks (Section 18.2, 18.3) [16] | Environment-specific risks | Environment-specific risks [5.2.2] |
| Secure coding practices [1] [6] [28] [29] | Use sound and proven secure coding practices to aid in reducing software defects introduced during implementation [1] [6] [28] [29] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] Implementation [6.2.1, 6.2.2] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| SAFECode [23] | Use secure coding practices: minimize unsafe function use; use latest compiler toolset; use static and dynamic analysis tools; manual code review; validate input and output; use anti-cross-site scripting libraries; use canonical data formats; avoid string concatenation for dynamic SQL; eliminate weak cryptography; use logging and tracing [23] [28] [29] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| Source code review for security vulnerabilities [1] [6] | Perform source code review using static code analysis tools, metric analysis, and manual review to minimize implementation-level security bugs [1] [6] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| | | Metrics analysis | Metrics analysis [3.2.1, 3.2.2, 3.2.3] |
| Secure software construction: | Common vulnerabilities; code construction (security principles, | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |

| References | Brief Description | Coding Categories | Applicable Knowledge Units |
|---|---|---|---|
| SwACBK [14] | coding standards and practices) [14] [28] [29] | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| Security engineering, coding: Anderson [16] | Sandboxing, proof-carrying code (Section 4.2.9) [16] | Proof-carrying code | Proof-carrying code [5.1.4, 6.2.1] |
| BSIMM domain, SSDL Touchpoints [2] | Code review: code review tools; development of customized rules; profiles for tool use by different roles; manual analysis; tracking/measuring results [2] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| | | Metrics analysis | Metrics analysis [3.2.1, 3.2.2, 3.2.3] |
| OWASP SAMM, verification [3] | Code review: security requirements review checklists; code-level vulnerabilities; intensive high-risk code reviews; automation; code analysis integrated with SDLC; language/application specific risks; code release gates [3] | Code inspections | Code inspections [5.1.4, 6.2.1] Risk management [2.2.3, 2.2.4, 2.3.3] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| Phase 3 implementation: Microsoft SDL [4] | Specify tools (code analysis, compilers); develop coding checklists; enforce banned functions; static analysis; user guidance; usage scenarios; many detailed practices [4] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [6.3.1, 6.3.2, 6.3.3, 6.3.4] Tools [6.1.1, 6.1.2] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |
| | | Environment-specific risks | Environment-specific risks [5.1.1] Risk management [2.3.3, 2.3.4] |
| Phase 4 verification: Microsoft SDL [4] | Secure code review (depth based on most at-risk components); defined exit criteria [4] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] Risk management [2.2.3, 2.2.4, 2.3.3] |
| *Computing Curriculum Series* [10] | Foundations of information security as part of programming fundamentals and secure programming [10] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| *Computing Curriculum Series* [10] | Robust and security enhanced programming [10] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| *SwA Pocket Guide* [17] | Improper input validation; improper encoding or escaping of output; cleartext transmission of sensitive information; failure to constrain operations within bounds of a memory buffer; failure to control generation of code (code injection); download of | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |

| References | Brief Description | Coding Categories | Applicable Knowledge Units |
|---|---|---|---|
| | code without integrity check; improper resource shutdown or release; improper initialization; use of broken or risk cryptographic algorithm; insecure permission assignment for critical resource; use of insufficiently random values [17] | | |
| ISC$^2$ CSSLP [22] | Declarative vs. programmatic security; common software vulnerabilities and countermeasures; defensive coding practices; exception management; configuration management; build environment; code peer review; code analysis; anti-tampering techniques; interface coding [22] | Assurance coding standards | Assurance coding standards [5.1.4, 6.2.1] |
| | | Code inspections | Code inspections [5.1.4, 6.2.1] |
| | | Coding security checklists | Coding security checklists [5.1.4, 6.2.1] |

*Table 5: Testing Practices*

| References | Brief Description | Testing Categories | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Security requirements/ penetration testing<br>Security unit testing (white box)<br>Security integration testing<br>Security functional testing<br>Code coverage analysis<br>Black box security tools<br>Fuzz testing | Security testing, all categories [3.1.1, 3.1.2, 3.3.1, 3.3.2, 4.1.4, 5.1.2, 5.2.1, 5.3.2, 6.2.1, 6.2.3, 6.3.5] |
| Unique aspects of software security testing [1] [6] | Understand the differences between software security testing and traditional software testing and plan how best to address these (including thinking like an attacker and emphasizing how to exercise what the software should not do) [1] [6] | Security requirements/ penetration testing<br>Security integration testing<br>Security functional testing<br>Black box security tools | Testing methods [6.2.1] |
| Functional test cases for security [1] [6] | Construct meaningful functional test cases (using a range of techniques) that demonstrate the software's adherence to its functional requirements, including its security requirements (positive requirements) [1] [6] | Security requirements/ penetration testing<br>Security functional testing<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |
| Risk-based test cases for security [1] [6] | Develop risk-based test cases (using, for example, misuse/abuse cases, attack patterns, or threat modeling) that exercise common mistakes, suspected software weaknesses, and mitigations intended to reduce or eliminate risks to ensure they cannot be circumvented (negative requirements) [1] [6] | Security requirements/ penetration testing<br>Security functional testing<br>Black box security tools | Risk-based test cases/threat modeling [3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.2.3, 2.2.4, 2.3.3] |
| Test cases using a range of security test strategies [1] [6] | Use a complement of testing strategies, including white box testing (based on deep knowledge of the source code), black box testing (focusing on the software's externally visible behavior), and penetration testing (identifying and targeting specific vulnerabilities at the system level) [1] [6] | Security requirements/ penetration testing<br>Security unit testing (white box)<br>Security integration testing<br>Security functional testing<br>Code coverage analysis<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |
| Assurance for CMMI, engineering [18] | Establish and maintain validation and verification procedures and criteria for the assurance of selected work products; analyze results of assurance validation and verification activities [18] | Security requirements/ penetration testing<br>Black box security tools | Validation [3.1.1, 3.1.2, 3.3.1, 3.3.2, 6.2.1]<br>Testing methods [6.2.1, 6.3.5] |
| Secure software verification, validation, and evaluation (V, V, and E): SwACBK [14] | Assurance cases; testing; dynamic analysis; static analysis; measurement; third party V, V, and E; tool assurance [14] | Security requirements/ penetration testing<br>Black box security tools | Assurance cases [3.1.1, 3.1.2]<br>Measurement [3.2.1, 3.2.2, 3.2.3]<br>Testing methods [6.2.1, 6.3.5] |
| BSIMM domain, | Security testing: integrate security in | Security | QA processes [1.2.2] |

| References | Brief Description | Testing Categories | Applicable Knowledge Units |
|---|---|---|---|
| SSDL Touchpoints [2] | standard QA processes; black box security tools; fuzz testing; risk-driven white box testing, application of attack model; code coverage analysis; focuses on vulnerabilities in construction [2] | requirements/penetration testing<br>Security unit testing (white box)<br>Fuzz testing<br>Code coverage analysis | Risk management [2.2.3, 2.2.4, 2.3.3]<br>Testing methods [6.2.1, 6.3.5] |
| OWASP SAMM, verification [3] | Security testing: test cases derived from security requirements; penetration testing; automation; security testing integrated with SDLC; language/application specific risks; test release gates [3] | Security requirements/penetration testing<br>Security functional testing<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |
| SAFECode [23] | Fuzz testing, penetration testing, third party assessment; use automated testing tools (eight listed) | Fuzz testing | Testing methods [6.2.1, 6.3.5] |
| Phase 4 verification: Microsoft SDL [4] | Security testing: ensure CAI of software and data; mitigate threat model threats; minimize vulnerabilities; test planning; fuzz testing; penetration testing; vulnerability regression testing; data flow testing; replay testing; input validation testing; privacy testing [4]<br>Security push: team-wide focus on threat model updates, re-evaluate attack surface, code review, testing, documentation [4] | Security requirements/penetration testing<br>Fuzz testing<br>Black box security tools | Threat model [3.1.2, 3.3.1, 3.3.2]<br>Testing methods [6.2.1, 6.3.5] |
| Construction: Bishop [07] | Unit testing [Section 18.2.2.3] [07] | Security unit testing (white box)<br>Code coverage analysis | Testing methods [6.2.1, 6.3.5] |
| Verification and Validation: Bishop [07] | Testing: security testing, integration and system testing [Sections 18.2.2.4, 19.3.3] [07]<br><br>Formal verification [Sections 19.2.4.3, 20.1, 20.3, 20.4] [07] | Security requirements/penetration testing<br>Security integration testing<br>Security functional testing<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |
| Security engineering, testing: Anderson [16] | Security testing: code, white box testing, architecture and implementation faults (Section 26.2.2.1) [16] | Security requirements/penetration testing<br>Security unit testing (white box)<br>Code coverage analysis | Testing methods [6.2.1, 6.3.5] |
| *Computing Curriculum Series* [9] | Software quality assurance including inspection of code and testing [9] | Security requirements/ penetration testing | Testing methods [6.2.1, 6.3.5] |
| *Computing Curriculum Series* [11] | Testing and quality assurance [11] | Security requirements/ penetration testing<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |
| *SwA Pocket Guide* [17] | Improper input validation; improper encoding or escaping of output; cleartext transmission of sensitive information; cross-site request forgery; error message information leak; failure to constrain operations within bounds of a memory buffer; improper resource shutdown or | Security requirements/ penetration testing<br>Black box security tools | Testing methods [6.2.1, 6.3.5] |

| References | Brief Description | Testing Categories | Applicable Knowledge Units |
|---|---|---|---|
| | release; improper initialization; incorrect calculation; use of broken or risk cryptographic algorithm; insecure permission assignment for critical resource; use of insufficiently random values [17] | | |
| ISC$^2$ CSSLP [22] | Testing for security QA (functional, security, environment, bug tracking, attack surface validation) [22] | Security requirements testing<br>Security functional testing<br>Security unit testing (white box) | Testing methods [6.2.1, 6.3.5] |
| | Test types (penetration, fuzzing, scanning, simulation, cryptographic validation) [22]<br>Impact assessment and corrective action [22]<br>Standards for software QA [22]<br>Regression testing [22] | Penetration testing<br>Fuzz testing | Penetration testing [5.1.2, 5.2.1, 5.3.2]<br>Impact assessment [4.1.4] |

*Table 6: Analysis of Software and Services in Static and Operational Contexts*

| Subject Matter | Analysis Artifacts | Analysis Objectives | Technologies and Methods | Applicable Knowledge Units |
|---|---|---|---|---|
| Software | Architectures, requirements, specifications, designs, code, documentation, test results, threat environment, analysis tool output | Assure software security through verification of required functional and security behavior and absence of malicious content | Reverse engineering through structuring and abstraction, correctness verification, static and dynamic analysis, inspection and review, security techniques and standards, risk management, assurance auditing | Verification/risk management [3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4]<br>Reverse engineering [6.3.1, 6.3.2, 6.3.3, 6.3.4] |
| Services | Service agreements, test results, operational and threat environments, operational performance | Assure service security through validation of required functional and security behavior and absence of malicious behavior | Testing, operational monitoring, security techniques and standards, risk management, assurance auditing | Validation/risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2, 6.2.1, 6.4.1, 6.4.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4]<br>Testing methods [6.2.1, 6.3.5] |
| Data | Databases, access methods and controls | Assure data security through validation of access controls and security properties | Operational monitoring, security techniques and standards | Operational monitoring, security techniques and standards [5.2.3, 7.2.1, 7.2.2, 7.2.3, 7.2.4]<br>Validation [3.1.1, 3.1.2, 3.3.1, 3.3.2] |
| Networks | Architectures, requirements, specifications, designs, test results, threat environment, operational monitoring tools | Assure network security through validation of security capabilities and operational performance | Flow analysis, operational monitoring, simulation, statistical analysis, failure analysis, security techniques and standards, risk management, assurance auditing | Flow analysis, operational monitoring, simulation, statistical analysis, failure analysis, security techniques and standards, risk management, assurance auditing [5.2.2, 7.2.1, 7.2.2, 7.2.3, 7.2.4 ]<br>Validation/risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 7.1.1] |
| Humans | Qualifications training, job performance, management | Assure human performance in achieving operational and security goals | Training programs, performance monitoring, insider threat analysis | Threat analysis [3.1.2, 3.3.1, 3.3.2] |
| Operations | Operational performance, threat | Assure operational integrity and | Operational monitoring, threat | Operational monitoring, threat |

| Subject Matter | Analysis Artifacts | Analysis Objectives | Technologies and Methods | Applicable Knowledge Units |
|---|---|---|---|---|
| | environment, intrusion detection, issues and problems, new requirements, maintenance, updates and patches | continuity through network and system management | tracking, vulnerability elimination, maintenance planning and implementation, continuity planning and preparation, risk management | tracking, vulnerability elimination, maintenance planning and implementation, continuity planning and preparation, risk management [5.2.2, 5.2.3, 4.1.2, 4.1.4, 7.2.1, 7.2.2, 7.2.3, 7.2.4 ]<br>Threat tracking/vulnerability elimination/risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4]<br>Continuity [3.2.4, 4.1.4, 7.3.1, 7.3.2]<br>Incremental development [6.2.1]<br>User training [7.1.3]<br>Maintenance [7.2.3] |

*Table 7: Assembly, Evolution, and Deployment*

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Incremental development and evolution | Incremental development and evolution [1.1.1, 1.1.2, 1.1.3, 6.2.1] |
| | | System integration | System integration [6.2.1] |
| | | User training | User training [7.1.3] |
| | | Maintenance and patching | Maintenance and patching [6.2.3] |
| | | System monitoring and management | System monitoring and management [5.2.1, 5.2.2, 7.2.1, 7.2.2, 7.2.3, 7.2.4, 7.3.1, 7.3.2] |
| Analysis of existing artifacts [Table 6] | Analysis of architectures, requirements, specifications, designs, code, test results, threat environment, vulnerabilities, security properties, service agreements, data access controls, monitoring results, training, job performance | Technologies and methods<br>• Threat, vulnerability, and security assessment<br>• Requirements, specifications, architecture, documentation, and code analysis<br>• Flow and service analysis<br>• Reverse engineering<br>• Verification and testing of security and functionality<br>• Application of standards and practices<br>Applied to<br>• software, services, data, networks, humans, and operations<br>For system types<br>• systems, systems-of-systems, distributed systems, SOA and cloud systems, infrastructure systems, embedded systems | Threat, vulnerability, and security assessment [3.1.2, 3.3.1, 3.3.2]<br>Service agreements [6.4.1, 6.4.2]<br>Artifact analysis [7.2.4]<br>Operational monitoring [7.2.1, 7.2.2]<br>Requirements, specifications, architecture, documentation, and code analysis [6.3.2, 6.3.2, 6.3.3, 6.3.4, 6.3.5]<br>Flow and service analysis [6,2,1, 6.3.1, 6.3.4]<br>Reverse engineering and analysis [6.3.1, 6.3.2, 6.3.3, 6.3.4]<br>Verification and testing [6.2.1, 6.3.5] |
| Analysis technologies [Table 6] | Reverse engineering, structuring, function abstraction, correctness verification, flow analysis, test design and evaluation, statistical analysis, capabilities and limitations of static and dynamic analysis and monitoring tools and intrusion detection tools, risk management, assurance auditing | Technologies and methods<br>• Threat, vulnerability, and security assessment<br>• Requirements, | Threat, vulnerability, and security assessment [3.1.2, 3.3.1, 3.3.2]<br>Verification/risk management/ assurance auditing [3.1.1, 3.1.2, 3.3.1, |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | | specifications, architecture, documentation, and code analysis<br>• Flow and service analysis<br>• Reverse engineering<br>• Verification and testing of security and functionality<br>• Application of standards and practices<br><br>Applied to<br>• software, services, data, networks, humans, and operations<br><br>For system types<br>• systems, systems-of-systems, distributed systems, SOA and cloud systems, infrastructure systems, embedded systems | 3.3.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4]<br>Analysis technologies [all of 6 and 7] |
| SAFECode [23] | Documentation for administrators' security configuration settings and the security and usability implications of those settings [23] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | System monitoring and management | System monitoring [7.2.1, 7.2.2, 7.2.3] |
| | | User training | User training [7.1.3] |
| ISC$^2$ CSSLP [22] | Software acceptance [22]<br><br>Pre-release or pre-deployment: completion criteria, risk acceptance, documentation [22] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | Post-release: verification and validation; independent testing [22] | System monitoring and management | Risk acceptance [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br>Risk acceptance [2.2.3, 2.2.4, 2.3.3]<br>Acceptance [6.3.5, 6.3.6, 6.4.1, 6.4.2] |
| SAFECode [24] | Ensure the presence of the software supply chain (supplier sourcing, product development and testing, product delivery) integrity controls that derive from security and integrity principles: chain of custody, least privilege access, separation of duties, tamper resistance and evidence; persistent protection, compliance management, code testing and verification [24] | Incremental development and evolution | Supply chain assurance [5.1.4, 6.3.5, 6.3.4, 6.4.1, 6.4.2]<br>Compliance [4.3.1, 4.3.1, 4.3.3] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| SAFECode [23] | Code integrity and handling: keep source code in well-protected source code control systems; verify chain of custody for the origin of software changes; protect against code tampering (active, at rest, in transit); monitor and analyze event and audit logs; sign code to verify integrity and authenticity; resolve bugs promptly and continuously | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1]<br>Project management [4.2.1, 4.2.2] |
| Security during software maintenance: JMU [6] | Particularly security-oriented procedures/ techniques for use when revising/enhancing legacy software [6] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | Maintenance and patching | Maintenance [7.2.3] |
| Incident management: Build Security In (BSI) [26] | Role of (computer security incident response team) CSIRT in SDLC, CSIRT definition, incident management [26] | System monitoring and management | System monitoring and management [7.2.1, 7.2.3, 7.2.4, 7.3.1, 7.3.2] |
| Deployment and operations: BSI [26] | Plan, Do, Check, Act, Integrating Security and IT, Prioritizing IT Controls, Navigating the Security Process Landscape [26] | Incremental development and evolution | Incremental development and evolution [5.1.4]<br>Project management [4.2.1, 4.2.2]<br>Process landscape [1.2.1, 1.2.2] |
| Secure software sustainment: SwACBK [14] | Monitoring for situational awareness (sensing); analysis; response management; update assurance cases [14] | User training | Assurance cases [3.1.2, 6.3.6]<br>User training [7.1.3] |
| | | System monitoring and management | Operational monitoring [7.2.1, 7.2.2, 7.3.1, 7.3.2] |
| BSIMM domain, deployment [2] | Penetration testing: focuses on outside/in testing, vulnerabilities in final configuration; supports defect management and mitigation [2] | Incremental development and evolution<br>System integration<br>User training<br>System monitoring and management | Incremental development and evolution [5.1.4]<br>System integration [6.2.1]<br>User training [7.1.3]<br>System monitoring and management [7.2.1, 7.2.3, 7.2.4, 7.3.1, 7.3.2]<br>Testing [6.2.1, 6.3.5] |
| | Software environment: OS and platform patching; web application firewalls; installation and configuration documentation; application monitoring; change management; code signing [2] | | |
| | Configuration management and vulnerability management: patching and updating applications; version control; defect tracking and remediation; incident handling [2] | | |
| OWASP SAMM, deployment [3] | Vulnerability management: incident response plan/team/process; security issue disclosure; root cause analysis; incident metrics<br>Environment hardening: install upgrades; patch management; monitor configuration; validate environment against best practices; audit<br>Operational enablement: application alert procedures; change management; operational security | Maintenance and patching | Process [1.2.2]<br>Maintenance [7.2.3] |
| | | System monitoring and management | System monitoring and management [7.2.1, 7.2.3, 7.2.4, 7.3.1, 7.3.2] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | procedures; code signing; audit [3] | | |
| Release: Microsoft SDL [4] | Final security review; final privacy review; response planning (vulnerability discovery; zero-day exploits); release permission dependent upon completing the defined SDLC process; respond to vulnerabilities/issues as they arise | Incremental development and evolution | Incremental development and evolution [5.1.4,6.2.1] SDLC [1.1.2, 1.2.1, 1.2.2] |
| | | System integration | System integration [6.2.1] |
| Legacy systems: BSI [26] | Assessing risk, COTS software security considerations [26] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] Assessing risk [2.2.1, 2.2.2, 2.3.1, 2.3.2, 3.1.2, 3.3.1, 3.3.2] |
| | | System integration | System integration [6.2.1] |
| | | Maintenance and patching | Maintenance [7.2.3] |
| Assembly and evolution: BSI [26] | Application firewalls and proxies, patch management, web service integration [26] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | System integration | System integration [6.2.1] |
| | | Maintenance and patching | Maintenance [ 7.2.3] |
| Operation and maintenance: Bishop [7] | Auditing (Chapter 14) [7] Intrusion detection (Chapter 23) [7] | Maintenance and patching | Maintenance [7.2.3] |
| | | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] |
| Attack and defense: JMU [6] | Attack and defense during operation [6] | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] |
| *Computing Curriculum Series* [10] | Security operations considerations in professional practice; might be more closely associated with deployment of software [10] | Maintenance and patching | Maintenance [7.2.3] |
| | | User training | User training [7.1.3] |
| | | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] |
| ISC$^2$ CSSLP [22] | Deployment, operations, maintenance, disposal [22] Installation and deployment: bootstrapping (key generation, access management); configuration management [22] | Systems integration<br><br>System monitoring and management | Systems integration [6.2.1]<br>Operational monitoring [7.2.1, 7.2.2]<br>User training [7.1.3] |
| | Operations and maintenance: monitoring, incident management; problem management; patching [22] End-of-life policies [22] | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] User training [7.1.3] |
| | | Maintenance and patching | Maintenance [7.2.3] |

*Table 8: Risk Mitigation Strategies for System Complexity and Scale*

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | Systems integration | System integration [6.2.1] |
| | | User training | User training [7.1.3] |
| | | Maintenance and patching | Maintenance and patching [7.2.3] |
| | | System monitoring and management | System monitoring and management [7.2.1, 7.2.2] |
| Tackle known interface vulnerabilities first [1] | With systems having more interfaces to less trusted systems, developers should concentrate first on known interface vulnerabilities such as those in web services. [1] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | System integration | System integration [6.2.1] |
| Conduct end-to-end analysis of cross-system work processes [1] | With increasing complexity, vulnerability analysis of individual systems is not sufficient. The security analysis of work processes that cross multiple systems has to consider the risks for those processes (including end-to-end analysis) as well as the risks that each work process creates for the systems that support it. Security analysis has to consider a wider spectrum of errors. [1] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] Processes [1.2.1] Risk analysis [2.2.1, 2.2.2, 2.3.1, 2.3.2, 3.1.2, 3.3.1, 3.3.2] |
| | | System integration | System integration [6.2.1] |
| Attend to containing and recovering from failures [1] [6] | Assume the existence of discrepancies of some form, whether in systems, operations, or users, during the execution of work processes, particularly as usage evolves. Give increased attention to containment and recovery from failures. These should be considered in the context of business continuity analysis. [1] [6] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] Processes [1.2.1] Continuity [3.2.4, 4.1.4, 7.3.1, 7.3.2] |
| | | System integration | System integration [6.2.1] |
| Explore failure analysis and mitigation to deal with complexity [1] | The multiplicity of systems and the increasing number of possible error states arising from their interactions can overwhelm analysis or generate too many point solutions that mitigate narrowly specified events. Explore how security could take advantage of a consolidated failure analysis and mitigation effort. [1] | System integration | System integration [6.2.1] |
| | | Maintenance and patching | Maintenance [7.2.3] |
| | | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] |
| Coordinate security efforts across organizational groups [1] | Security is typically treated as a separate concern, with responsibility often assigned to independent parts of the organization. It is not unusual to find that an organization's development, operational, and business groups are tackling common problems with little coordination, or that some security problems have fallen through the cracks. This | User training | User training [7.1.3] |
| | | System monitoring and management | Operational monitoring [7.2.1, 7.2.2] Vulnerability analysis [3.1.2, 3.3.1, 3.3.2] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | separation is even more problematic as the scope and scale of systems expand. Vulnerability analysis and mitigations should be integrated across organization units, users, technology, systems, and operations. [1] | | |
| Certification and accreditation: JMU [6] | Mainly concerning systems and software but some attention given to personnel certification and meeting CNSS requirements [6] | Incremental development and evolution | Incremental development and evolution [5.1.4, 6.2.1] |
| | | User training | User training [7.1.3] |

*Table 9: Governance and Management Practices*

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Bracketed numbers refer to the references at the end of Appendix B. | | Business case | Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4, 7.1.1, 7.1.2] |
| | | Risk management | Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1] |
| | | Awareness | Awareness [part of all KUs; 5.1.2]<br>Business case (awareness for business leaders) [4.1.1, 4.1.2, 4.1.3, 4.1.4] |
| | | Training, education, certification | Training, education, certification [1.2.1, 7.1.3] |
| | | Project management (process management) | Project management [1.1.1, 1.2.2, 4.2.1, 4.2.2, 7.1.1, 7.1.2] |
| | | SwA practices integrated with SDLC | SwA practices/process [1.1.1, 1.1.2, 1.1.3, 1.2.1, 1.2.2, 6.2.1, 7.1.2] |
| | | Transition and adoption | Adoption [1.1.1, 1.1.2, 1.2.1, 1.2.2, 7.1.1] |
| | | Measurement | Measurement [1.2.2, 3.2.1, 3.2.2, 3.2.3, 6.3.6] |
| | | Ethics | Ethics [5.3.1] |
| | | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3, 6.3.6] |
| | | Evaluation (systems, software, people) | Evaluation [3.1.2, 3.3.1, 3.3.2, 5.2.4, 6.1.1, 6.3.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5, 6.3.6] |
| | | Acquisition | Acquisition [1.1.3, 1.2.1, 1.2.2, 6.4.1, 6.4.2] |
| Business case [5] | Present economic and additional arguments that describe the need for software assurance and the impact of its absence: ROI, cost-benefit, models, vendor measurement, SIDD [5, other BSI] | Business case | Vendor measurement [3.2.1, 3.2.2, 3.2.3]<br>Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4, 7.1.1] |
| Phase 1 requirements: Microsoft SDL [4] | Determine if developmental and support costs for improving security and privacy are consistent with business needs [4] | Business case | Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4, 7.1.1] |
| CMU—Heinz [21] | Analytical tools for calculating the costs and benefits of investment security decisions and how to calculate the return on investments in a hands-on setting; commercially | Business case, risk management | Risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.2.1, 2.2.2, 2.2.3, |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | available tools for risk management; introduction to vulnerability management; risk aversion and insurance [21] | | 2.2.4]<br>Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4, 7.1.1] |
| Risk-based definition of adequate security [1] | Identify ways to determine what constitutes adequate security practice based on risk management, established levels of risk tolerance, and risk assessment [1] | Risk management | Risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1]<br>Adequate security practice [2.2.3, 2.3.4] |
| Continuous risk management framework [1] [6] | Put a continuous, business-driven risk management framework in place and periodically assess for acceptable and unacceptable levels of risk throughout the SDLC [1] [6] | Risk management | Risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2]<br>Risk management [2.2.1, 2.2.2, 2.2.3, 2.2.4, 7.1.1] |
| OWASP SAMM, governance [3] | Strategy and metrics: business risk profile; assurance roadmap; data/application classification; align investment with asset value | Risk management, business case, SwA project management | Risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1]<br>Risk profile [2.2.1, 2.2.2]<br>Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4]<br>Project management [4.2.1, 4.2.2] |
| CMU—Heinz [21] | The role of market and competition on security provision and then some of the key causes of market failure, namely externalities, how various policy tools can be applied to mitigate market failure, key laws and regulation on product liability, and security standards [21] | Risk management, compliance | Risk management [3.1.1, 3.1.2, 3.3.1, 3.3.2, 7.1.1]<br>Risk management [2.2.1, 2.2.2, 2.2.3, 2.2.4]<br>Compliance [4.3.1, 4.3.1, 4.3.3] |
| Software security as a cultural norm [1] [6] | Recognize that being aware of security and understanding the importance of addressing security during software development needs to be a cultural norm (beliefs, behaviors, capabilities, actions) [1] [6] | Awareness | Awareness [5.1.2] |
| Characteristics of software security at the governance/ management level [1] | Engage leaders and stakeholders to better appreciate and understand the characteristics and actions necessary to address software security as governance and management concerns, and the consequences of not doing so [1] | Awareness | Business case [4.1.1, 4.1.2, 4.1.3, 4.1.4] |
| Assurance for CMMI, training [18] | Establish and maintain strategic assurance training needs of organization [18] | Training | Training [7.1.3] |
| Pre-SDL requirements, security training: Microsoft SDL [4] | In security requirements, secure design, threat modeling, secure coding, security testing, privacy [4] | Training | Training [7.1.3] |
| BSIMM domain, governance [2] | Training of software developers and architects [2] | Training | Training [7.1.3] |
| OWASP SAMM, governance [3] | Education and guidance of software developers, all personnel on secure SDLC based on role; comprehensive training and certification [3] | Training | Training [7.1.3, 1.2.1] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| Assurance for CMMI, project management [18] | Define project objectives for assurance; define the scope of assurance for the product or service; identify and analyze assurance-related project risks; ensure that adequate resources to execute assurance plans are provided; monitor assurance risk; select suppliers based on an evaluation of their ability to meet assurance requirements and criteria; document supplier agreement for assurance; define and select risk management strategy due to vulnerabilities and safety hazards [18] | SwA project management | Project management [4.2.1, 4.2.2]<br>Risk analysis [2.2.1, 2.2.2, 2.2.4,, 2.3.1, 2.3.2, 7.1.1] |
| Security engineering, management: Anderson [16] | Project management: security projects, risk management, organizational issues, process assurance (Section 25.2, 25.5, 25.6, 26.2.3) [16]<br>Security evaluation: relying on third party evaluation, common criteria evaluation, protection profile (Section 26.3) [16] | SwA project management | Project management [4.2.1, 4.2.2]<br>Process [1.1.1, 1.1.2, 1.1.3, 1.2.1]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 7.1.1]<br>Evaluation [3.1.2, 3.3.1, 3.3.2, 6.3.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5, 6.3.6] |
| Secure software project management: SwACBK [14] | Start up; scoping; risk management; SDLC selection; configuration management; software QA for security [14] | SwA project management | Project management [4.2.1, 4.2.2]<br>Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4. 7.1.1] |
| BSIMM domain, governance [2] | Strategy and metrics: planning; assigning roles and responsibilities; identifying software security goals; determining budgets; identifying metrics and gates [2] | SwA project management; measurement | Project management [4.2.1, 4.2.2]<br>Measurement [3.2.1, 3.2.2, 3.2.3, 6.3.6] |
| Software security practices integrated with SDLC [1] [6] | Provide recommendations for inserting security practices into the SDLC as part of traditional project management activities, including the use of defined security touchpoints at each life-cycle phase [1] [6] | Integrate with SDLC | Project management [4.2.1, 4.2.2]<br>Management [1.1.1, 1.2.2, 7.1.1] |
| Secure software processes: SwACBK [14] | Heavyweight; lightweight; legacy upgrade; introducing and improving security practices as part of SDLC [14] | Integrate with SDLC | Processes [1.1.1, 1.1.2, 1.1.3, 1.2.1, 1.2.2, 7.1.2] |
| BSIMM domain, governance [2] | Standards and requirements: recommend COTS; develop standards for security controls (authentication, input validation, etc.); develop standards for technologies in use; create standards review board [2] | Integrate with SDLC | Requirements [6.2.1] |
| Enterprise software security framework [1] | Establish a framework and roadmap for addressing software security as an enterprise-wide undertaking and identify some of the pitfalls and | Adoption: framework | Adoption [1.2.2, 7.1.1] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| | barriers to tackle head on [1] | | |
| Assurance for CMMI, process management [18] | Establish and maintain: description of assurance context and objectives for the organization; organizational processes to achieve assurance business objectives; tailoring criteria and guidelines for assurance in organization's set of standard processes; assurance of organization's work environment based on the organization's work environment standards [18] | Adoption: process management | Processes [1.1.1, 1.1.2, 1.2.1, 1.2.2, 7.1.2] |
| Software security included in software development measurement process [1] [6] | Determine how to include security as part of a software development measurement process, including suggested process and product measures, and implement, track, and report such measures [1] [6] | Measurement | Measurement [3.2.1, 3.2.2, 3.2.3, ] Measurement [6.3.6] Process [1.2.2, 7.1.2] |
| Assurance for CMMI, support [18] | Define and improve project assurance measures; store assurance measures appropriately [18] | Measurement | Measurement [3.2.1, 3.2.2, 3.2.3] Measurement [6.3.6] |
| Ethics, law, and governance: SwACBK [14] | Ethics, laws, regulations, policy, standards [14] | Ethics | Ethics [5.3.1] |
| | | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] |
| Ethics, legal, policy, and standards considerations: JMU [6] | Overview of ethical and compliance sources; requirements and issues [6] | Ethics | Ethics [5.3.1] |
| | | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] |
| BSIMM domain, governance [2] | Compliance and policy: identifying controls for compliance with specific regulations, such as PCI, HIPAA; developing contractual controls (SLAs); setting organizational software security policy; auditing against policy [2] | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] Measurement [6.3.6] |
| OWASP SAMM, governance [3] | Policy and compliance: build policies and standards for security and compliance; establish audit practice; create compliance gates for projects | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] Measurement [6.3.6] |
| Audit [25] | Standards and requirements for software and system security audits [25] | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] Measurement [6.3.6] |
| CMU—Heinz [21] | Strategies of various countries to gain control and secure communication and information flows in cyberspace, concepts of security, why information and network security become a public policy issue; critical choices policy makers face in their efforts to develop strategies to secure cyberspace, how national governments deal with the limited reach of territorially bound rules and regulation in a network of networks without bounds [21] | Compliance | Compliance [4.3.1, 4.3.2, 4.3.3] Measurement [6.3.6] |
| Certification and accreditation: JMU [6] | Mainly concerning systems and software but some attention given to personnel certification and meeting CNSS requirements [6] | Evaluation | Measurement [6.3.6] |

| References | Brief Description | Category | Applicable Knowledge Units |
|---|---|---|---|
| *Computing Curriculum Series* [8] | Defense security implementation and management: the organizational activities associated with the selection, procurement, implementation and management of security processes and technologies for IT infrastructures and applications [8] | Acquisition | Process [1.2.1, 1.2.2] Acquisition [6.4.1, 6.4.2] |
| Acquisition [26] | Life-cycle considerations; trusted vendors; systems of systems; government acquisition; business (commercial) acquisition [26] | Acquisition | Process [1.1.4] Acquisition [6.4.1, 6.4.2] |
| Acquiring secure software: SwACBK [14] | Planning (need, requirements, decision to acquire, risk management); software reuse; RFP process (acquirer); supplier response (software architecture; assurance plan); source selection (evaluation criteria); contract negotiation [14] | Acquisition | Process [1.1.4] Risk management [2.1.1, 2.1.2, 2.1.3, 2.2.1, 2.2.2, 2.2.3, 2.2.4, 7.1.1] Acquisition [6.4.1, 6.4.2] |

## Appendix B Bibliography


**[1]**
Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Professional, 2008.

**[2]**
McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model BSIMM v1.0*. http://www.bsi-mm.com/ (March 2008).

**[3]**
OpenSAMM Project. *Software Assurance Maturity Model (SAMM) v1.0*. http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model (2009).

**[4]**
Microsoft. *Security Development Life Cycle, Version 4.1*. http://www.microsoft.com/sdl (2009). The SDL addresses, in separate sections, practices for product software vs. practices for internal line of business (LOB) software. Only those for product software were reviewed.

**[5]**
Mead, Nancy R.; Allen, Julia H.; Conklin, Arthur W.; Drommi, Antonio; Harrison, John; Ingalsbe, Jeff; Rainey, James; & Shoemaker, Dan. *Making the Business Case for Software Assurance* (CMU/SEI-2009-SR-001). Carnegie Mellon University Software Engineering Institute, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09sr001.cfm

**[6]**
James Madison University. *Secure Software Engineering Curriculum*. Includes independent input from Sam Redwine. http://www.cs.jmu.edu/SSS/grad_program/curriculum.html (Accessed 2009).

**[7]**
Bishop, Matt. *Computer Security: Art and Science*. Addison-Wesley Professional, 2003.

**[8]**[20]
The Association for Computing Machinery (ACM); The Association for Information Systems (AIS); & The Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS). "Computing Curricula 2005." *Computing Curriculum Series*. http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/CC2005-March06Final.pdf (2005).

**[9]**[20]
IEEE-CS & ACM. "Computer Engineering." *Computing Curriculum Series.* http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/CCCE-FinalReport-2004Dec12-Final.pdf (2004).

**[10]**
ACM & IEEE-CS. "Computer Science Curriculum 2008: An Interim Revision of CS 2001." *Computing Curriculum Series*. http://www.acm.org/education/curricula/ComputerScience2008.pdf (2008).

**[11]** [20]
ACM & IEEE-CS. "Computing Curricula: Information Technology Volume." *Computing Curriculum Series*. http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/IT_draft_curriculum.pdf (2008).

**[12]**
Gorgone John T.; Davis, Gordon B.; Valacich, Joseph S.; Topi, Heikki; Feinstein, David L.; & Longenecker, Herbert E., Jr. "IS 2002: Model Curriculum and Guidelines for Undergraduate Programs in Information Systems." *Computing Curriculum Series*. Association for Information Systems. http://192.245.222.212:8009/IS2002Doc/Main_Frame.htm (2002).

**[13]**
IEEE-CS & ACM. "Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." *Computing Curriculum Series*. http://sites.computer.org/ccse/SE2004Volume.pdf (2004).
[Not referenced in any practice; included for completeness.]

**[14]**
Department of Homeland Security (DHS) Software Assurance (SwA) Workforce Education and Training Working Group. *Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software.* Edited by Samuel T. Redwine, Jr. https://buildsecurityin.us-cert.gov/daisy/bsi/940-BSI/version/default/part/AttachmentData/data/CurriculumGuideToTheCBK.pdf (2007).

[20] Membership is required to access this document.

**[15]**
Dougherty, Chad; Sayre, Kirk; Seacord, Robert; Svoboda, David; & Togashi, Kazuya. *Secure Design Patterns* (CMU/SEI-2009-TR-101, ESC-TR-2009-010). Carnegie Mellon University Software Engineering Institute, 2009.
http://www.sei.cmu.edu/library/abstracts/reports/09tr010.cfm

**[16]**
Anderson, Ross J. *Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd Edition*. Wiley, 2008.

**[17]**
DHS SwA Processes & Practices Working Group & Technology, Tools and Product Evaluation Working Group. "Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses," Table 1. *Software Assurance Pocket Guide Series: Development, Volume 2, Version 1.3*. Department of Homeland Security Software Assurance Program, March 2009.
https://buildsecurityin.us-cert.gov/swa/downloads/KeyPracticesMWV13_02AM091013.pdf

Other guides in this series include

- "Contract Language for Secure Software"
- "Software Supply Chain Risk Management and Due Diligence"
- "Risk-Based Software Security Testing"
- "Contract Language for Integrating Software Security in the Acquisition Life Cycle"

Guides are available at https://buildsecurityin.us-cert.gov/swa/pocket_guide_series.html.

**[18]**
Material adapted from

- Croll, Paul & Moss, Michele. "Leveraging CMMs and Standards for Assurance." Paper presented at the Systems & Software Technology Conference (SSTC) 2008, Las Vegas, Nevada, Track 6, Wednesday, April 30, 2008.
- Croll, Paul & Moss, Michele. "Leveraging the CMMI To Address Assurance – Benchmarking Assurance Practices and Managing Assurance Risks." Paper presented at the NDIA CMMI Technology Conference 2008, Denver, Colorado, November 19, 2008.
- Moss, Michele & Nadworny, Margaret. "Update on the Assurance for CMMI Practices." Department of Homeland Security (DHS) Software Assurance Working Group, December 4, 2008.
- DHS SwA. *Summary of Assurance for CMMI Efforts*. https://buildsecurityin.us-cert.gov/swa/downloads/Assurance_for_CMMI_Pilot_version_March_2009.pdf (2009).
- DHW SwA. *Processes and Practices Working Group*.
- https://buildsecurityin.us-cert.gov/swa/procwg.html (2009).

**[19]**
Teumin, David J. *Industrial Network Security*. The Instrumentation, Systems, and Automation Society (ISA), 2004 (ISBN 1556178743).

**[20]**
Yasar, Ansar-Ul-Haque; Preuveneers, Davy; Berbers, Yolande; & Bhatti, Ghasan. "Best Practices for Software Security: An Overview." *Proceedings of the 12th IEEE International Multitopic Conference*. Bahria University Karachi, Sindh Pakistan, December 2008.

**[21]**
Carnegie Mellon relevant courses (URLs accessed in 2009)

- Master of Science in Information Security Technology and Management (MSISTM), The Information Networking Institute, Carnegie Mellon University. http://www.ini.cmu.edu/degrees/pgh_msistm/index.html
- Courses in the Information Networking Institute, Carnegie Mellon University. http://www.ini.cmu.edu/degrees/pgh_msistm/courses.html
- Courses in Master of Science in Information Security Policy and Management, H. John Heinz III College, Carnegie Mellon University. http://www.heinz.cmu.edu/school-of-information-systems-and-management/information-security-policy-management-msispm/index.aspx
- Curriculum of the Information Security Policy & Management (MSISPM) Program, School of Information Systems & Management, H. John Heinz III College, Carnegie Mellon University. http://www.heinz.cmu.edu/school-of-information-systems-and-management/information-security-policy-management-msispm/curriculum/index.aspx
- List of courses in the Information Security Policy & Management (MSISPM) Program, School of Information Systems & Management, H. John Heinz III College, Carnegie Mellon University. http://www.heinz.cmu.edu/school-of-information-systems-and-management/information-security-policy-management-msispm/curriculum/course-information-BAK/index.aspx
- Course descriptions in the H. John Heinz III College, Carnegie Mellon University. http://www.heinz.cmu.edu/index.aspx
- Carnegie Mellon University School of Computer Science (SCS) International Software Research Institute (ISRI) master's programs http://mse.isri.cmu.edu/software-engineering/web1-Programs/index.html

[Content is embedded in courses. Not obvious where it resides.]

**[22][21]**
(ISC)$^2$. *Certified Secure Software Lifecycle Professional (CSSLP) Candidate Information Bulletin.* http://www.isc2.org/cib/default.aspx (June 27, 2009).

**[23]**
Software Assurance Forum for Excellence in Code (SAFECode). *Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today.* Edited by Stacy Simpson. http://www.safecode.org/publications/SAFECode_Dev_Practices1108.pdf (2008).

[21] Registration is required to download this document.

**[24]**
Software Assurance Forum for Excellence in Code (SAFECode). *The Software Supply Chain Integrity Framework: Defining Risks and Responsibilities for Security Software in the Global Supply Chain.* Edited by Stacy Simpson.
http://www.safecode.org/publications/SAFECode_Supply_Chain0709.pdf (2009).

**[25]**
Review comments from Yair Levy at Nova Southeastern University; October 26, 2009.

**[26]**
DHS SwA. *Build Security In*. https://buildsecurityin.us-cert.gov/daisy/adm-bsi/home.html (2009).

**[27]**
Viega, John & McGraw, Gary. *Building Secure Software*. Addison-Wesley, 2002.

**[28]**
Seacord, Robert. *The CERT C Secure Coding Standard*. Addison-Wesley, 2008.
http://www.informit.com/store/product.aspx?isbn=0321563212.

**[29]**
CERT and Oracle. *The CERT Oracle Secure Coding Standard for Java*.
https://www.securecoding.cert.org/confluence/display/java/The+CERT+Oracle+Secure+Coding+Standard+for+Java (2010).

## Appendix B Acronyms

**ACM**
Association for Computing Machinery

**API**
application programming interface

**ARA**
architectural risk analysis

**BoK**
body of knowledge

**BSI**
Build Security In

**BSIMM**
Build Security In Maturity Model

**CAI**
confidentiality, availability, and integrity

**CMM**
Capability Maturity Model

**CMMI**
Capability Maturity Model Integration

**CMU**
Carnegie Mellon University

**CNSS**
Committee of National Security Systems

**COTS**
commercial-off-the-shelf

**CSIRT**
Computer Security Incident Response Team

**DHS**
Department of Homeland Security

**DoS**
denial of service

**HIPAA**
Health Insurance Portability and Accountability Act

**IEEE**
Institute of Electrical and Electronics Engineers

**IEEE-CS**
Institute of Electrical and Electronics Engineers Computer Society

**INI**
Information Networking Institute

**ISA**
Instrumentation, Systems, and Automation Society

**ISC$^2$ CSSLP**
International Information Systems Security Certification Consortium, Inc. Certified Secure Software Lifecycle Professional

**ISRI**
International Software Research Institute

**IT**
information technology

**JMU**
James Madison University

**KU**
knowledge unit

**LOB**
line of business

**MSISPM**
Master of Science in Information Security and Policy Management

**MSISTM**
Master of Science in Information Security Technology and Management

**NDIA**
National Defense Industrial Association

**OS**
operating system

**OWASP**
Open Web Application Security Project

**PCI**
Payment Card Industry

**PD3+C**
Privacy by Design, Privacy by Default, Privacy in Deployment, and Communications

**QA**
quality assurance

**RFP**
request for proposal

**ROI**
return on investment

**SAFECode**
Software Assurance Forum for Excellence in Code

**SAMM**
Software Assurance Maturity Model

**SCS**
School of Computer of Science

**SD3+C**
Secure by Design, Secure by Default, Secure in Deployment, and Communications

**SDL**
software development lifecycle (Microsoft)

**SDLC**
software development life cycle

**SEI**
Software Engineering Institute

**SIDD**
security investment decision dashboard

**SLA**
service level agreement

**SOA**
service-oriented architecture

**SQL**
structured query language

**SQUARE**
Security Quality Requirements Engineering

**SSDL**
software security development life cycle

**SSTC**
Systems & Software Technology Conference

**STRIDE**
Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, Elevation of privilege

**SwA**
software assurance

**SwACBK**
Software Assurance Curriculum Body of Knowledge

**V, V, and E**
verification, validation, and evaluation

# Appendix C: Interview Questionnaire Summary

The questionnaire and responses are presented here in their original state and have been edited only to correct misspellings.

### Interview Questions

The purpose of this questionnaire is to assist in the development of a curriculum model for a Master of Software Assurance degree (a project led by the Software Engineering Institute and sponsored by the Department of Homeland Security).

For the purpose of this project, software assurance is defined as: "…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" (from CNSS 4009 IA Glossary).

We appreciate your willingness to help in our effort to better prepare software assurance and security professionals for the workplace. Please respond to the questions below and feel free to elaborate on your answers with additional comments.

1. Assume you are interviewing to fill a position for a specialist in software assurance or software security. Please rate, with a checkmark, each of the following capabilities you might consider in hiring such an individual. We would like you to provide responses using two ways of rating these capabilities:
   - how you currently rate such capabilities for a prospective employee regardless of their academic background and experience (Current)
   - how you would rate a prospective employee that had a master's degree with a focus on software assurance (MSwA). Of course, you may decide to rate them the same.

*<Note: These are totally different positions – there is virtually no overlap for them. It's like saying you can hire a plumber or a baker. Where is the common ground? A specialist in software security is a FUNCTIONAL expert – e.g., a product manager or developer in identity management or in cryptography or in database security. Software assurance experts are not product specialists – they are orthogonal. The skill sets have virtually no overlap in our organization.>*

*Also, you are creating a dichotomy between "hiring a person for a software assurance positions" and "hiring a MSwA for an assurance position." There is no such dichotomy. I'd hire anybody qualified for an assurance position whether or not they had a master's in it. Also, there are few "assurance positions" as we define them that are interchangeable. Assurance to us includes everything from security evaluators to program managers to ethical hackers – all very different skill sets.*

*Please rate, with a checkmark, each of the following capabilities you might consider in hiring such an individual. We would like you to provide responses using two ways of rating these capabilities:*

**Capability:**

Think like an attacker in understanding and analyzing intruder motivations and methods, threat environments, and vulnerabilities for new and existing software systems. Be capable of introducing appropriate security technologies and methods, and verifying software functionality for conformance to requirements and absence of malicious content.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X | | |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | | X<br>X | | |

Comments:

- These could be separate capabilities – and would have been rated differently if they weren't combined.
- Both offensive and defensive skills, knowledge, and abilities are now required on your team of developers or accreditors/certifiers. The ability to hack, embed-n-wait, actively bring down a system or system-of-systems is the "attacker" skill set – you would necessarily pair these folk with "defenders". You are mixing both in this paragraph to the left – without using the terms explicitly.
- It is imperative that someone that is going to protect something knows just how to attack it. This not only means a great facility with the "tools" but with a demonstrated ability to think "out of the box" with both attack vectors and "defense in depth" solutions.
- This is an essential skill for pen testers and security professionals.
- Thinking like an attacker is a critical component of "software assurance." The total set of competencies listed under "capability" would only be something we'd expect of a very senior candidate – with or without a MSwA degree. Verifying software functionality for conformance to requirements in general goes well beyond the security definition of software assurance – one would normally focus only on resistance to attack and freedom from malicious content except in the case of specific security mechanisms.
- To think like an attacker is of more value in high value targets such as classified software development or systems. When software is exploited it seems to be the same type of attacks,

such as buffer overflow. If you're attacking a network you scan for the types of software is being used on the network and on the nodes. From that point, the attacker can diagnose how they want to continue. Basic security measures, such as virus protection, patches, and standard desktop security measures can prevent almost all attacks.

- The second part I believe is more important. The ability to properly diagnose and implement the appropriate security measures is paramount to proper security implementation. Making sure the software does what it is supposed to do and that it's free of malicious code is a good first step.
- An MSwA should have training in thinking like an attacker. However, some people are just naturally better at this than others. Expertise in this area will come from seeing the actual attacks that happen.
- My rationale is that, if someone had MSwA on their resume, they would be expected to have this capability, while I would not expect it in the average programmer.
- I can't comment since sentence one and sentence two are totally different skills. Security technologies and methods exist as feature sets independent of "think like an attacker." I expect EVERY developer to think like a hacker whether or not they work on security features. Unfortunately, they do not – we have to train them to do it. By falsely creating a "hire a security person or an assurance person," it creates a dichotomy that does not exist because the skills sets are non-comparable and the positions are not interchangeable. Also, every development manager has to be able to "verify software functionality for conformance to requirements." I would never, ever have a "assurance person" validate every piece of code – it would be unworkable and unscalable. Lastly, "absence of malicious content" is not possible to prevent and code review will not catch these. Let's not kid ourselves.
- While it would be desirable to employ someone today with these capabilities, except in limited cases, this still remains absent in most curricula. While I would expect the capabilities in someone with a MSwA, it would actually be nice if there was at least some exposure to the skills in all curricula aimed at educating developers whether at the Bachelors or Masters level.
- 90% of security problems are exploits of well-known vulnerabilities or gaps in process. Creative attack analysis is icing on the cake, not the foundation.
- I have found that this skill perhaps above all others is the most applicable to build credible threat models. Most curriculums in IT teach people how to build systems/programs, but very few look deeply at what happens when they fail or are induced to fail.
- The answers to all of these items are relative to new employees fresh out of an undergraduate program or an MSwA program. The assumption is that they have no experience beyond internship.

**Capability:**

Identify and integrate software assurance practices and technologies into all phases of the software development life cycle. Perform risk and tradeoff analysis subject to project constraints to optimize security properties and functional correctness in new system development and legacy system evolution.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X-<br>X-<br>X-<br>X-<br>X-<br>X- | X-<br>X-<br>X-<br>X-<br>X-<br>X- | X-<br>X- | | |
| MSwA | X-<br>X-<br>X-<br>X-<br>X-<br>X-<br>X-<br>X- | X-<br>X-<br>X- | X- | | |

Comments:

- This is where it would probably become necessary to segregate MSwA graduates into those that concentrated on security implementations, detection and protections and those that were more oriented in the software design and development areas (assurance).
- Most of times we observe this type of skills come from practices. While individuals educated on these topics have a head start, proven hands-on experience is the differentiator.
- I rate this list of capabilities “nice to have” for the same reason that I questioned the set of competencies listed above: we have hundreds of people who work in various aspects of software security and software assurance, but only a few bring this total set of capabilities. And I find it inconceivable that any newly graduated MSwA holder would actually have such capabilities although I suppose that a “no hire” candidate might believe he or she did.
- For a current employee, I would look for a solid software quality engineering background. The MSwA should be trained in software quality engineering principles. You have to understand software quality practices before you can apply security practices and technologies.
- Being able to perform risk and tradeoff analysis is very important. Training in risk frameworks related to software assurance would be beneficial.
- As software security begins to be more of a reflex than an afterthought in the development lifecycle this capability may be your most important ability.
- Near term budgets for security will be limited and shrinking as the project moves along. The ability to understand the value of what needs to be protected and negotiating a minimum level security is key.
- I am not sure what you mean by security property.

- Also, when you talk about lifecycle phases, I wonder if you mean to include source code control. This is a capability that not many programmers have before being trained on the job.
- Again, sentence 1 and 2 should be evaluated separately. I'd expect any person who worked as a development manager or release manager to be able to do 1 – but realize, as assurance is a "process overlay" has to be imposed through, say, release management. You can't hire one person who knows assurance and expect anything positive from that unless that person is in program management. Risk and tradeoff analysis I expect every developer to do since security is ONE factor in ALL development projects even if it is not a security feature or security product. So is performance, for example. Any organization with a decent process will include security as a boilerplate part of their design documents.
- The earlier practices and tech are introduced, the stronger the probability that they are incorporated.
- This is core requirement for software assurance.

**Capability:**

Apply analysis methods and technologies, such as assurance cases, to measure and validate the effectiveness of practices in achieving desired system security and functionality.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X | | |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X | | | |

Comments:

- This would be necessary to not only keep up with the designs currently in use but also to try to keep ahead of emerging threats.
- Note that, at the current state of the art, an intent to apply "assurance case" to software security would qualify a candidate as a "no hire" since assurance case is still a theoretical approach that, to our knowledge (and that of John Goodenough) has not been applied in the real world. I would expect a MSwA to understand threat modeling and other proven approaches to security analysis and to understand why no real world project should attempt to apply assurance case at its current state of maturity.
- Need to be able to measure your security processes and practices.
- I would expect an MSwA to also be able to apply "abuse" cases as described by McGraw in *Software Security*.
- I don't know what an assurance case is – we do not use that language. Assurance is Not Equal to "desired system security and functionality." The capability is only important in people who work in assurance positions and even then, only limited positions – e.g., assurance also includes positions such as security evaluators and vulnerability analysts – while the capability is important for a "pure assurance" or "program management" function, there are many positions for which it is not relevant nor required. Nobody would hire this capability for a product manager or development manager, for example.

- Rated as “Desired” without the MSwA as I would expect most graduates to be able to apply test cases and would hope they could at least have some fundamental understanding of security needs.
- If this means “testing” then yes!

**Capability:**

Apply assurance technologies and methods across a spectrum of systems, including systems-of-systems, network systems, infrastructure and embedded systems, and service-oriented and cloud computing environments.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X | X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | | X |
| MSwA | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X | | X |

Comments:

- Although it would be nice to have this as a requirement for an MSwA graduate, this level of sophistication is really a doctoral level skill-set.
- Only a very experienced and capable candidate would be expected to bring this breadth of capability, and only a very few positions require it. Again, I'd be dubious if a newly graduated MSwA claimed this level of competence.
- An area that has proven to lie outside the line of sight of most security measures is the large multi-agency and in the future more cloud computing environments. Knowing you must do more than give lip service or send a security policy to your customers and business partners that use your systems. You must ensure they're taking the necessary measure to protect their gateways into your systems or software.
- My rationale is that I would not know if this was covered in the MSwA curriculum but I would hope that it would be.
- I think it's an unrealistic requirement for a development organization. Cloud computing has SECURITY implications that are not merely assurance implications. And I doubt you'd ever hire one security person who would have all the above areas of responsibility. Information assurance, maybe, but software assurance – this capability is not relevant.
- This may be one of the more difficult concepts to grasp (i.e., the effect on security – and functionality – of the inter-operability of component systems), but is critical to understanding the assurance question.
- Unclear how this is different (if that is the intent) than the item above.
- Expectation that this is learned via on the job experiences, but having academic back ground is useful.

**Capability:**

Communicate compelling business and technical arguments on the value of software assurance to executives, project managers, and peers, sufficient to catalyze adoption of assurance practices.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | | X |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | | | |

Comments:

- Communication skills, marketing skills, budget case development skills – being able to wedge your request into a regularly funded and regularly monitored, successful activity – nice to have -- but the Master Degree grad surely should be technically gifted first and foremost – and have the marketable skills to apply to the IT infrastructure and IT solution set before this.
- This ability would apply to both concentrations, assurance and security. However, this ability is something that will usually be developed when working on the management elements of projects involving security. This would be a good candidate for a course that was geared to the MBA groups specializing in information systems and infrastructure.
- This is a highly desirable skill but not necessarily a requirement.
- Many of our security program managers are required to “sell” software security assurance to development groups and their management. I’d worry that someone with a new MSwA degree might try to make the case for assurance in a way that would be too theoretical and disconnected from real-world business value.
- Very important as security is again not a reflex. More often than not, you will need to fight for a limited amount of resources.
- In order for software assurance to be taken seriously within an organization, it is necessary for senior management to be supportive of the role prior to establishing the position of software assurance manager. It is much more important that the candidate can fulfill the other position requirements.

- And, someone needs to be able to do ECONOMIC analysis, not just theory of good development. Showing which assurance measures have the highest payoff the fastest is critical. An assurance expert without business (and economic) analysis skills is worthless.
- I would hope by now all developers would understand the arguments, but sadly know this is likely not true. While I would expect someone with a MSwA to be able to express these arguments, I would think in most cases it would be to project managers, peers, and one category that I think you left out but is highly important, the client executives requesting the software. I think it would be important for a MSwA holder to be able to articulate the arguments to company or IT executives, but I think generally that responsibility would fall to more senior IT management.
- In theory this should be Required but the audiences listed here – execs, PMs, etc. – each need a very specific presentation.
- Building business cases are the key to making a SA program effective. The key inside the business case is articulating the technical implications of the threat into the business impacts potentially avoided by the SA controls.

**Capability:**

Apply software assurance practices in the elicitation, analysis, and specification of software requirements.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X | X | |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X | X | X | |

Comments:

- Once again, this would be beneficial for an MSwA that had a concentration in assurance.
- I don't understand what this means.
- The ability to identify security needs early on in projects can save money. This trend will lead to security becoming a reflex.
- Don't understand this. Software assurance is a process. Unless the people building features understand assurance, an "expert" will not help them and will not make a positive difference. If you have 1000 developers per software assurance expert, the assurance expert will not succeed unless he/she gets the 1000 developers to do the right thing, and that requires process and education, not hands on "specification of software requirements."
- This would seem fundamental to the education of an MSwA.
- See item #2 above. The sooner good design is documented, vetted, and assessed against a set of assurance practices, the easier it is to implement them.

**Capability:**

Apply software assurance practices in the development of a software architecture and the design of software components and modules.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X | | |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X | | | |

Comments:

- For the assurance specialist, this would be mandatory as requirements, design and code reviews could help remove these as issues before they get into the design thereby saving a lot of effort later.
- This is a crucial skill. I would not hire a security professional who doesn't have experience building systems the correct way.
- We certainly require the design of secure software components and modules, but a newly graduated MSwA is going to start out under the supervision of experienced project leaders, not be sent off to design new architectures on his or her own.
- What does assurance have to do with an architecture? Design of components I get, but again, ANY developer should be able to do this.
- Same as previous comment (i.e., fundamental to the curriculum).
- This sounds like the implementation phase of the requirements phase above. Security a z-dimensional design aspect like performance or dynamic binding behavior – you want to incorporate it into components so that the profile in terms of interaction with other modules is deterministic and (in this case) the strength of security commensurate with risk and need.

**Capability:**

Apply software assurance practices in the implementation and testing of software modules.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X | | |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X | | X | |

Comments:

- Test engineering is a very specialized capability and usually takes training and a lot of experience.
- Regardless of whether the individual is a tester or a developer this is a vital skill.
- Familiarity with effective tools and practices that support the implementation and testing of secure software is a definite plus.
- This would be desirable for a QA person, very desirable. Not relevant for a “security person” who is a product manager and not necessarily relevant for a developer. Would never use a MSWA as a tester. Would use ethical hackers for that but MSWA would not have hacking skills.
- Same as previous comment (i.e., fundamental to the curriculum).
- Consistent with my assertions above.
- Expectation that day to day execution is done by development groups in the SLDC which was implemented as part of the SA program.

**Capability:**

Apply software assurance practices in the verification and validation of software artifacts, both developed and acquired.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X | | X |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X | X<br>X | | X |

Comments:

- This too is test oriented and would be difficult to teach without the additional ability to try it out in real life. If this were to be included, this would be an area that could be used to solicit grant money from corporations where the students were given money to develop procedures, tools and practices to assist the corporation on specific projects.
- This description is not sufficiently specific for me to evaluate. Our processes focus on software and its design and on artifacts that directly reflect the design and implementation of the software. If verification and validation are oriented toward documents that are not an integral part of a security-oriented commercial development process, they are ineffective and thus of no value to us.
- You must be kidding. “Validation of software artifacts?” What does that mean? Making sure design documentation is complete? That’s a release management function. You could never possibly hire enough assurance people at a large development organization to do that work and the effort would be doomed to fail. If assurance is not part of your development process, it will absolutely FAIL as an overlay function.
- Same as previous comment (i.e., fundamental to the curriculum).
- Unclear how this differs from the above especially the “testing.”

**Capability:**

Apply software assurance practices in assembly, evolution, and operations of software.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X | | X<br>X |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X | X<br>X | | X<br>X |

Comments:

- I don't understand this capability description. If it refers to operational security expertise, that's nice to have, but we hire primarily for people who can contribute to secure design, development, and testing.
- I would expect the MSwA to be able to both design and compare secure installation and configuration alternatives.
- Not relevant at all. "Evolution?" What is that? Bug fixing? Creation of patch sets? Not sure.
- Same as previous comment (i.e., fundamental to the curriculum).
- Assembly? That's the 2 items above. Evolution? That's back to requirements, only it's "new" requirements and that's OK. Operations? OK, that's different --- and it deserves its own line. Super important! There should be operational detective controls built into solutions.

**Capability:**

Understand software assurance activities associated with the procurement and acquisition of software and software-intensive systems.

| | Required | Desired | Nice To Have | Not Important | No Opinion |
|---|---|---|---|---|---|
| Current | X<br>X<br>X<br>X | X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X | | X |
| MSwA | X<br>X<br>X<br>X<br>X<br>X<br>X | X<br>X<br>X<br>X | X | | X |

Comments:

- Procurement, acquisition, legacy systems, GOTS, COTS, integration, and glue are key to current systems development. These activities are a likely vector for unintended consequences as well as exploits. An appreciation of this domain is necessary.
- When we acquire software for integration into our products, we apply the same security requirements as we do for software that we develop.
- The MSwA should understand what questions to ask of vendors for software assurance.
- I think people are actually inherently better at analyzing what could go wrong at a vendor than they are at what might happen in their own software shops.
- Not relevant to us at all.
- Same as previous comment (i.e., fundamental to the curriculum).
- Procured or home-built, the process should be the same.
- This is an emerging area of requirements for the commercial sector and will likely become more important over time, but is not essential right now.

15. Are there other capabilities or issues that are important to your organization when hiring a software assurance or software security professional?

- Ability to apply software assurance in spiral, agile and incremental development models
- Ability to contribute software assurance in preparation of proposals in response to RFPs
- Awareness and understanding of various standards.
- You would want the Master's grad to have an understanding of the legal framework of the clients/customers – like HIPAA, FISMA, NISPOM, NIST, et al. This governs privacy concerns, security concerns, etc and governs, ultimately – customer requirements for Certification and Accreditation, etc.
- You would obviously want to see *nix OS/Windows* OS experience, Cisco/Juniper/et al switch/router experience; Oracle/Sybase/*base/SQL experience; web app/client app experience; web server/mail server experience, et al.
- You have stressed – via the structure of your questions – the importance of life-cycle experience across a large-scale project – I would think that the successful Masters Student would either come with that experience or be exposed to the equivalent. Individuals new to the workforce may need additional years of experience to achieve mid- to advanced-level positions.
- It would be interesting to develop real-world attack scenarios as well as real-world defense scenarios where a student would be able to live through a cyber attack/challenge with the current type of attack/defense experiences of the last 2-3 years. Place at their disposal kiddie scripts, malware, virus types, DOS tools, etc. Make them think like an attacker and give them the tools to bring down a system – if that is possible. You are training them in the world of the potential adversary.
- It would be nice to have a curriculum for non-specialists as well – Everyone who works with software needs to know everyday good practices to avoid common security pitfalls and test for vulnerabilities.
- Soft skills
  - Ethics and social responsibility
  - Technical and business writing
  - Leadership – to champion security and mentor others
- Technology infrastructure (computing, telecommunications, networks)
- Awareness in related IT operations services such as configuration management, change management, incident management, business continuity
- For an assurance student, it should be mandatory that they come from a background with extensive programming capability in at least one high order language (preferably C because that is still the main language of the telephone management infrastructure) and on multiple platforms (UNIX, Linux, etc.).
- We always look for individuals who strive to keep abreast of the current developments in the industry including new tools, libraries and attacks.

- The entire concept of the MSwA seems somewhat strange, and out of kilter with the way that real software development organizations operate. Software design, development and testing are done by development organizations, and we seek to have those organizations integrate security ("software assurance") into their work. We do this by providing program managers, developers, and testers, with training, tools, and guidance that are applicable to their work – to the way that software is actually built.
- We also have an organization that focuses on the development of training materials, tools, guidance, and processes that will help the development organizations do the security part of their job more effectively. This organization is almost exclusively staffed with security experts who have real-world development experience. So when they introduce a new process or requirement such as threat modeling, they either have a high degree of confidence that it will be implementable by development groups or they are able to design low-cost valid experiments to help them gain that confidence. The reference to assurance case in the "capability" table above suggests a program that will produce MSwA degree holders who know how secure development and security assurance "should" work based on textbook knowledge, but have not experienced for themselves real-world development or the ways that software can suffer from security failures.
- Our experience is that the best source of software security people is expert developers, testers, and program managers who have developed an interest in security and then supplemented their software expertise with real-world and theoretical understanding of security and assurance.
- Knowledge in the following disciplines:
  - Access Control
  - Application Development Security
  - Business Continuity and Disaster Recovery Planning
  - Cryptography
  - Information Security Governance and Risk Management
  - Legal, Regulations, Investigations and Compliance
  - Operations Security
  - Physical (Environmental) Security
  - Security Architecture and Design
  - Telecommunications and Network Security
- Since many of the security best practices are applied to a software development lifecycle, I would expect that the MSwA would have a good understanding of software quality engineering principles as well has familiarity with software quality frameworks (e.g., CMMI).
- Some background in project management should be required for a current employee and MSwA. You have to think about cost, schedule, and performance. Security, like quality, cost money. Risk analysis is also a factor since you don't want to spend $1000 protecting a $10 asset.
- All of the identified capability areas are important to software assurance. Even with software quality professionals, they all have some baselines training but then they end up specializing

in one area – e.g. requirements, risk analysis, configuration management. Being able to apply what is taught at a master's degree program is difficult to measure. I would hope the program would integrate many real examples of compromised systems to train the master's degree student in applying what they've learned. The more they are exposed to real examples, the better they will be able to apply what they've learned.

- A master's degree program should teach all of the capabilities identified in the survey. The real expertise, however, comes from experience.
- The ratings, which I provided above, are the same for Current and MSwA based upon the assumption that the position requires the set of capabilities, except for one capability that I note as desirable but not required for the reason stated. My point is that the position requirements are independent of the candidate. I would expect that the MSwA would meet all the requirements, whereas other candidates, without the MSwA degree, would be expected to fail many of the requirements. The list of capabilities should include familiarity with computer programming languages in general as well as specific working knowledge of one or more programming languages, preferably including those that are commonly deployed by the hiring organization. Another requirement should be knowledge of application monitoring, failure or error detection, response, recovery and reconstitution.
- Experience with computer operations and/or technical support is highly desirable. All too often, developers and testers have inadequate experience with production environments. It is crucial to understand how systems are deployed, operated and supported. The curriculum should include these areas.
- Your first question did not address secure coding practices explicitly. I would expect an MSwA to be completely conversant in the OWASP top 25 coding mistakes and be able to avoid them, correct them, and teach others to as well.
- You did not mention the ability to adopt standards. For example, it is well understood by security professionals that software developers should never write their own cryptography libraries, but most of them do not have a clue how to go about choosing one to incorporate into their systems. I would expect an MSwA to be good at this.
- Again, these positions are 100% different. Nobody would hire an assurance expert to run the identity management development group and I would likely not hire a cryptographer for a program manager role in assurance.
- This entire questionnaire assumes the presence of a "assurance expert" group that waves the magic wand over development – but not everybody has the Microsoft assurance model. Some of this think the only way to be successful is to embed assurance throughout development – [my company], for example, creates "security points of contact" within each development group that are boots on the ground. We (the assurance [people) train them but we are not the hiring managers – development hires people to be developers and they are then selected to be SPOCs. Ergo, assurance is a collateral duty not their main job and we would hire first for their main skill set.
- I understand the point of this questionnaire but I couldn't answer ma y questions because we are not organized like that and we embed assurance differently. A model where "the assurance expert" signs off on what developers do I believe is a fundamentally flawed model in development groups of any size. There will never be enough security police to make an

environment secure unless assurance is highly decentralized and highly embedded within development. An assurance expert can help do that but cannot exercise the amount of direct, feature by feature development oversight this questionnaire assumes.

- I believe you have covered the core competencies. The only other thing I could think of is the ability to research current sources and vectors of attack so as to keep current in understanding "Think like an attacker in understanding and analyzing intruder motivations and methods, threat environments, and vulnerabilities for new and existing software systems".
- I believe the applicant would need to be able to understand workflow processes that are in house, and adapt the software assurance activities to be seamless wherever possible throughout the work flow.
- The applicant would also need the ability to understand many different systems some airborne, some ground based and how the integration of those domains could be compromised.
- The installation of such systems requires at times temporary installations (hot swap methods) that must also undergo scrutiny.
- I believe some functions (or ways the functions are expressed or implemented) may make a system vulnerable. As such the applicant should be able to understand when such conflicts could potentially occur and how to mitigate accordingly.
- A requirement to make a system predictive or deterministic may conflict with a software assurance requirement. (e.g. because the system is deterministic, the expected behavior is known and may provide a system attribute that could become vulnerable to attacks.) The applicant should be able to understand such situations.
- I believe that for development of new systems (including those that have COTS) there should be some type of 'Plan Against Vulnerabilities' for the development that should be part of the or appended to the hazard analysis with resultant flow down of appropriate requirements to the system. The applicant should be able to provide such a plan as part of their responsibilities.
- The most important thing for a SA professional is to develop a plan that is
  1. *Implementable* with precision
  2. *Measurable* without Herculean effort but with precision
  3. *Clear* on what aspects – breadth and/or depth – are covered and NOT covered by the Plan.
- Too many SAs seek out the esoteric, complicated edge cases that make it seem like they are experts in the field. The fact is security is much more about straightforward, clear, easy-to-implement-and-measure elements. Driving to the more common use cases typically means driving back to mainstream development – and that's where many development teams would rather sidestep everything except the most bluntly obvious security designs and practices. I could fund a nice holiday party if I had a dime for every time I heard a development team say "We don't need to have credential protection on [our] web service because only our apps are hitting it and they are protected on the front end." Security through perceived obscurity is still alive and well – and needs to be managed down.

- Has to have deep understanding of the development process (but not how to code) and the internal and external influences that shape applications being delivered to customers. The delivery of complex applications over long delivery cycles with multiple iterative releases creates many complexities when implementing software assurance that are probably not apparent on the surface.
- It would be beneficial if a Software Assurance program could provide indications early in the development cycle of potential flaws and vulnerabilities in the system or application. The Software Assurance expert should understand design principles and system architecture, be able to identify potential threats to the system or application, the vectors that could be used to exploit a vulnerability, be able to specify security functional requirements for the system or application as a result of the analysis of threats and vectors, and have the ability to verify the application of controls has met the security functional requirements. The Software Assurance expert should have the ability to quantify and categorize risk in terms that application owners will readily understand and relate to.
- Ability to develop a repeatable, measurable program. Document the evidence and tests to ensure appropriate knowledge transfer and handle regression testing etc.
- In today's environment one would be expected to have a solid understanding of other computing topics such as networks and associated vulnerabilities, mobile communications, cloud computing, and COTS integration, not just isolated software and systems.

# Appendix D: Comparison to Other Programs

Authors of this report who are faculty at Embry-Riddle Aeronautical University and Monmouth University provided the following discussion.

### Comparison to Embry-Riddle Aeronautical University (ERAU)

The ERAU Master of Software Engineering (MSE) program currently requires 15 credits of the core courses (i.e., five courses at three credits each) and three credits of a capstone experience [ERAU 2010]. Students can select an additional 12 credits of elective courses from the software engineering (SE) electives list. The capstone experience can be either an individual research project or a practicum typically offered as a team project on development of a substantial software artifact. The most current departmental discussion led to a new proposal that an existing elective course on software quality and assurance (SE625) be moved to the core.

With permission from the MSE program coordinator, the students with strong computing background or an appropriate SE experience may take up to six credits of graduate technical electives outside the SE program. Such an option allows us to introduce two courses dedicated specifically to software assurance. Additionally, by modifying the selected course content, we could direct the program toward more assurance. Example courses that could be appropriate for such modifications are SE505, SE545, and SE575. Certainly there is room for some minor modification of all core courses to highlight the software assurance issues.

#### Required Core Courses (15 credits)

- SE 500 Software Engineering Discipline
- SE 510 Software Project Management
- SE 530 Software Requirements Engineering
- SE 555 Object-Oriented Software Construction
- SE 610 Software Systems Architecture and Design

#### Capstone Experience (3 credits)

- SE 697 Software Engineering Practicum *OR* SE690 Graduate Research Project

#### Elective Courses (12 credits)

- SE 505 Model Based Verification of Software
- SE 520 Formal Methods for Software Engineering
- SE 535 Graphical User Interface Design and Evaluation
- SE 545 Specification and Design of Real-Time Systems
- SE 550 Current Trends in Software Engineering
- SE 575 Software Safety
- SE 580 Software Process Definition and Modeling
- SE 585 Metrics and Statistical Methods for Software Engineering
- SE 590 Graduate Seminar

- SE 625 Software Quality Engineering and Assurance
- SE 565 Concurrent and Distributed Systems
- SE 655 Performance Analysis of Real-Time Systems
- SE 660 Formal Methods for Concurrent and Distributed Systems
- SE 699 Special Topics in Software Engineering

### Modifying Monmouth University's Master of Science in Software Engineering (MSSE) Program to Add a Software Assurance Track

If we were to modify the Monmouth University MSSE program to create a track in software assurance, we would most likely replace our current track in telecommunications with the software assurance track. The curriculum for the telecommunications track is made up of the following courses, all of which are three-credit courses:

#### Preparatory Courses

- CS 501B Program Development
- CS 503 Data Structures and Algorithms
- SE 504 Principles of Software Engineering
- SE 510 Object Oriented Analysis and Design
- SE 515 Disciplined Software Development (PSP)

#### Core Courses

- SE 561 Mathematical Foundations of Software Engineering
- SE 565 Software Systems Requirements
- SE 570 Software Systems Design
- SE 575 Software Verification, Validation and Maintenance
- SE 580 The Process of Software Engineering (CMMI, TSP and agile methods)

#### Telecommunications Track Courses

- SE 637 Wireless Communications
- SE 620 Network Software I
- SE 621 Network Software II

#### Telecommunications Track Elective Courses

Two courses chosen from among

- CS 514 Networks
- CS 526 Performance Evaluation
- CS 535 Telecommunication
- SE 610 Software Systems Security
- SE 611 Secure Web Services Design
- SE 638 Communications Systems
- CS 505 Operating System Concepts

### Practicum or Thesis

- SE 685A and SE 685B Software Practicum
- SE 690A and SE 690 B Thesis

To create a software assurance track, we would substitute it for the telecommunications track by simply replacing the three courses required by the telecommunications track with three to five courses that cover material in the MSwA2010 Body of Knowledge. We would ensure these new courses would not duplicate material in the core software engineering courses in the Monmouth University MSSE program. The structure of the software assurance track would look like the following. All courses would be three credits.

## Potential Software Assurance Track in Monmouth University's MSSE Program

### Preparatory Courses

- CS 501B Program Development
- CS 503 Data Structures and Algorithms
- SE 504 Principles of Software Engineering
- SE 510 Object Oriented Analysis and Design
- SE 515 Disciplined Software Development (PSP)

### Core Courses

- SE 561 Mathematical Foundations of Software Engineering
- SE 565 Software Systems Requirements
- SE 570 Software Systems Design
- SE 575 Software Verification, Validation and Maintenance
- SE 580 The Process of Software Engineering (CMMI, TSP and agile methods)

### Software Assurance Track Courses

- SE 610 Software System Security (Note this is an existing course currently used as an elective.)
- SwA 60X First Software Assurance Course (with content from the MSwA2010 BoK)
- SwA 60X Second Software Assurance Course (with content from the MSwA2010 BoK)
- SwA 60X Third Software Assurance Course (with content from the MSwA2010 BoK)

### Software Assurance Track Elective Courses

One course chosen from among

- SE 611 Secure Web Services Design
- CS 518 Fundamentals of Computer Security and Cryptography
- CS 528 Database and Transaction Security

### Practicum or Thesis

- SE 685A and SE 685B Software Practicum
- SE 690A and SE 690B Thesis

# Appendix E: Comparison of MSwA2010 Knowledge Units to GSwE2009 Core BoK Knowledge Units and Maturity Levels

Table 1 depicts a comparison of the MSwA2010 knowledge units with the BoK knowledge units in the GSwE2009 Core BoK. The purpose of the comparison is to determine where and to what degree there are differences in the two BoKs. This comparison is meant to highlight where the MSwA2010 project will need to develop curriculum courses and modules.

The column labeled *GSwE2009 BOK Coverage* uses the following notation:

- The letter notation refers to a knowledge area (KA) and a knowledge unit (KU) in the GSwE2009 Core BoK. For example, J1 refers to the KA/KU => Software Engineering Process/Process Implementation and Change.
- The term *no coverage* means there is no explicit reference to the MSwA2010 KU topic in the GSwE2009 Core BoK.
- The term *limited coverage* means there is little coverage of the MSwA2010 KU topic in the GSwE2009 Core BoK, and this topic is mostly outside the scope of the GSwE2009.
- The term *supplemented* means there is coverage of the MSwA2010 KU topic in the GSwE2009 Core BoK, but this coverage must be supplemented with additional material to reach the intent of the MSwA2010 KU.

Table 1 also compares the SwA BoK with maturity levels from *Software Security Engineering: A Guide for Project Managers* [Allen 2008] using the MSwA2010 curriculum. Maturity levels vary among knowledge units, with some knowledge units less mature than others. We feel it is important to teach students all the relevant knowledge in the field, even though some areas may be less mature, so that they know what is available.

The table uses the following maturity levels adapted from the *Software Security Engineering: A Guide for Project Managers* book for the MSwA2010 curriculum [Allen 2008]:

- L1: The content provides guidance for how to think about a topic for which there is no proven or widely accepted approach. The intent of the description is to raise awareness and aid the reader in thinking about the problem and candidate solutions. The content may also describe promising research results that may have been demonstrated in a constrained setting.
- L2: The content describes practices that are in early pilot use and are demonstrating some successful results.
- L3: The content describes practices that have been successfully deployed (mature) but are in limited use in industry or government organizations. They may be more broadly deployed in a particular market sector.

- L4: The content describes practices that have been successfully deployed and are in widespread use. Readers can start using these practices today with confidence. Experience reports and case studies are typically available.

*Table 1: Comparison of MSwA2010 BoK and GSwE2009 BoK*

| Breakdown of Topics | MSwA2010 Bloom Level | GSwE BOK Coverage | GSwE2009 Bloom Level | Maturity Level |
|---|---|---|---|---|
| 1. Assurance Across Life Cycles | | | | |
| 1.1. Software Life-Cycle Processes | | | | |
| New development | C | J1, J2 | C/AP | L4 |
| Integration, assembly, and deployment | C | J1, J2 | C/AP | L4 |
| Operation and evolution | C | B7, J1, J2 | C/AP | L4 |
| Acquisition, supply, and service | C | J1, J2 | C/AP | L3 |
| 1.2. Software Assurance Processes and Practices | | | | |
| Process and practice assessment | AP | J3 | AP | L3 |
| Software assurance integration into SDLC phases | AP | limited coverage | | L2/3 |
| 2. Risk Management | | | | |
| 2.1. Risk Management Concepts | | | | |
| Types and classification | C | I2 | AP | L4 |
| Probability, impact, severity | C | I2 | AP | L4 |
| Models, processes, metrics | C | I2 | AP | L3–metrics; L4 |
| 2.2. Risk Management Process | | | | |
| Identification | AP | I2 | AP | L4 |
| Analysis | AP | I2 | AP | L4 |
| Planning | AP | I2 | AP | L4 |
| Monitoring and management | AP | I2 | AP | L4 |
| 2.3. Software Assurance Risk Management | | | | |
| Vulnerability and threat identification | AP | no coverage | | L3 |
| Analysis of software assurance risks | AP | no coverage | | L3 |
| Software assurance risk mitigation | AP | no coverage | | L3 |
| Assessment of software assurance processes and practices | AP | limited coverage | | L2/3 |
| 3. Assurance Assessment | | | | |
| 3.1. Assurance Assessment Concepts | | | | |
| Baseline level of assurance; allowable tolerances, if quantitative | AP | no coverage | | L1 |
| Assessment methods | C | I4, K3 | C/AP | L2/3; L4–vulnerability assessments /scans |
| 3.2. Measurement for Assessing Assurance | | | | |
| Product and process measures by life-cycle phase | AP | J4 | AP | L1/2 |
| Other performance indicators that test for the baseline as defined in 3.1.1, by life-cycle phase | AP | limited coverage | | L1/2 |
| Measurement processes and frameworks | C | I6, J1, J4 | C/AP | L2/3 |
| Business survivability and operational continuity | AP | limited coverage | | L2 |

| Breakdown of Topics | MSwA2010 Bloom Level | GSwE BOK Coverage | GSwE2009 Bloom Level | Maturity Level |
|---|---|---|---|---|
| 3.3. Assurance Assessment Process (collect and report measures that demonstrate the baseline as defined in 3.1.1) | | | | |
| Comparison of selected measurements to the established baseline | AP | J4 | AP | L3 |
| Identification of out-of-tolerance variances | AP | no coverage | | L3 |
| 4. Assurance Management | | | | |
| 4.1. Making the Business Case for Assurance | | | | |
| Valuation and cost-benefit models; cost and loss avoidance; return on investment | AP | I7 | C | L3 |
| Risk analysis | C | limited coverage | | L3 |
| Compliance justification | C | no coverage | | L3 |
| Business impact/needs analysis | C | B1, B3, C3 | C/AP | L3 |
| 4.2. Managing Assurance | | | | |
| Project management across the life cycle | C | I1, I3 | AP | L3 |
| Integration of other knowledge units | AN | limited coverage | | L2/3 |
| 4.3. Compliance Considerations for Assurance | | | | |
| Laws and regulations | C | limited coverage | | L3 |
| Standards | C | limited coverage | | L3 |
| Policies | C | limited coverage | | L2/3 |
| 5. System Security Assurance | | | | |
| 5.1. For Newly Developed and Acquired Software for Diverse Systems | | | | |
| Security and safety aspect of computer-intensive critical infrastructure systems such as power, telecommunication, water, and air traffic control | K | no coverage | | L2 |
| Potential attack methods | C | no coverage | | L3 |
| Analysis of threats to software | AP | limited coverage | | L3 |
| Methods of defense | AP | limited coverage | | L3 |
| 5.2. For Diverse Operational (Existing) Systems | | | | |
| Historic and potential operational attack methods | C | limited coverage | | L4 |
| Analysis of threats to operational environments | AN | limited coverage | | L3 |
| Design of and plan for access control, privileges, and authentication | AP | limited coverage | | L3 |
| Security methods for physical and personnel environments | AP | no coverage | | L4 |
| 5.3. Ethics and Integrity in Creation, Acquisition, and Operation of Software Systems | | | | |
| Overview of ethics, code of ethics, and legal constraints | C | A1, A2 | C/AP | L4 |
| Computer attack case studies | C | no coverage | | L3 |
| 6. System Functionality Assurance | | | | |
| 6.1. Assurance Technology | | | | |

| Breakdown of Topics | MSwA2010 Bloom Level | GSwE BOK Coverage | GSwE2009 Bloom Level | Maturity Level |
|---|---|---|---|---|
| Technology evaluation | AN | limited coverage | | L3 |
| Technology improvement | AP | limited coverage | | L3 |
| 6.2. Assured Software Development | | | | |
| Development methods | AP | C-all, D-all, E-all, F-all + supple-mented | C/AP/AN | L2/3 |
| Quality attributes | C | C-all, D-4 + supple-mented | C/AP/AN | L3–depends on the property |
| Maintenance methods | AP | G-all + supple-mented | C/AP | L3 |
| 6.3. Assured Software Analytics | | | | |
| Systems analysis | AP | limited coverage | C/AP | L2–architectures; L3/4–networks, databases (identity management, access control) |
| Structural analysis | AP | limited coverage | C/AP/AN | L3 |
| Functional analysis | AP | limited coverage | C/AP/AN | L2/3 |
| Analysis of methods and tools | C | limited coverage | C/AP/AN | L3 |
| Testing for assurance | AN | limited coverage | C/AP/AN | L3 |
| Assurance evidence | AP | limited coverage | AP/AN | L2 |
| 6.4. Assurance in Acquisition | | | | |
| Assurance of acquired software | AP | limited coverage | | L2 |
| Assurance of software services | AP | limited coverage | | L3 |
| 7. System Operational Assurance | | | | |
| 7.1. Operational Procedures | | | | |
| Business objectives | C | limited coverage | | L3 |
| Assurance procedures | AP | limited coverage | | L3 |
| Assurance training | C | limited coverage | | L4 |
| 7.2. Operational Monitoring | | | | |
| Monitoring technology | C | limited coverage | | L4 |
| Operational evaluation | AP | limited coverage | | L4 |
| Operational maintenance | AP | G-all + supple- | | L3 |

| Breakdown of Topics | MSwA2010 Bloom Level | GSwE BOK Coverage | GSwE2009 Bloom Level | Maturity Level |
|---|---|---|---|---|
| | | mented | | |
| Malware analysis | AP | no coverage | | L2/3 |
| 7.3. System Control | | | | |
| Responses to adverse events | AN | D-4, F-3 + supple-mented | AP | L3/4 |
| Business survivability | AP | no coverage | | L3 |

# Appendix F: Course Descriptions for the MSwA2010 Curriculum

The following are course descriptions for nine courses that could compose an MSwA stand-alone program, as well as seven courses that could be added to an MSwE program. The knowledge units that each course should cover appear in parentheses by the course name. (Over the next several months we will be adding reference material for each course, including publicly available items like Build Security In (BSI) website articles, CrossTalk articles, books, standards, and other relevant materials such as CWE, CAPEC, and so on. These updates will be available at the web page for the Software Assurance Curriculum project: http://www.cert.org/mswa.)

### MSwA Stand-Alone Program (Nine Courses)

Assurance Management (2.1, 2.2, 2.3, 4.1, 4.2, 4.3)

Assurance Assessment (3.1, 3.2, 3.3, 6.4)[22]

System Operational Assurance (7.1, 7.2, 7.3)

System Security Assurance (5.1, 5.2, 5.3)

Assured Software Analytics (6.3)

Assured Software Development 1 (1.1, *1.2*, 6.1, 6.2 [requirements])[23]

Assured Software Development 2 (6.1, 6.2 [specification, design])

Assured Software Development 3 (6.2 [code, test, verification, validation])

Software Assurance Capstone Experience

### MSwA Courses Added to MSwE Program (Seven Courses)

Assurance Management (*1.2*, 2.1, 2.2, 2.3, 4.1, 4.2, 4.3)

System Operational Assurance (**3.1, 3.2, 3.3, 6.4,** 7.1, 7.2, 7.3)[24]

System Security Assurance (5.1, 5.2, 5.3)

Assured Software Analytics (6.3)

Assured Software Development 1 (1.1, 6.1, 6.2 [requirements, specification, design])

Assured Software Development 2 (6.2 [code, test, verification, validation])[25]

---

[22] This course is not present in the MSwA Courses Added to MSwE program.

[23] The 1.2 knowledge unit, italicized, is different in Assured Development 1 in the stand-alone program and Assurance Management in the MSwA Courses Added to MSwE program.

[24] The bolded knowledge units are not covered at the same Bloom's level as in the stand-alone program.

Software Assurance Capstone Experience

## MSwA Stand-Alone Program (Nine Courses)

### Course: Assurance Management (2.1, 2.2, 2.3, 4.1, 4.2, 4.3)

#### Catalog Description

This course covers the fundamentals of software and system assurance management, including making the business case for assurance; planning and managing development projects that include assurance practices; compliance with laws, regulations, and standards, and policies related to assurance; and risk assessment, identification, analysis, mitigation, and monitoring for assurance.

#### Expected Outcomes

After completing this course, students will be able to

1. make a business case for assurance
2. understand how to add assurance considerations and practices as part of normal project management activities
3. identify, analyze, and select assurance practices that are relevant for a specific software development or acquisition project
4. understand laws, regulations, standards, and policies that are relevant to assurance
5. understand basic risk management concepts
6. identify, analyze, plan for, mitigate, and monitor assurance risks
7. identify risks arising from vulnerabilities and threats
8. determine assurance processes and practices that mitigate risks

---

[25] Condensed versions of Assured Software Development 1, 2, and 3 are in the stand-alone program.

### Course: Assurance Assessment (3.1, 3.2, 3.3, 6.4)

#### Catalog Description

This course covers the fundamentals of establishing a required level of software and system assurance. It also covers applying methods and determining measures to assess if the required level of assurance has been achieved. Topics include assessment methods; defining product and process measures and other performance indicators; measurement processes and frameworks; performance indicators for business survivability and continuity; and comparing selected measures to determine if the software/system meets its required level of assurance. These fundamentals are applied to newly developed software and systems, as well as to the acquisition of software and services.

#### Expected Outcomes

After completing this course, students will be able to

1. specify a required level of assurance for a system
2. understand how to use a range of assessment methods, including requirements validation, risk analysis, threat analysis, vulnerability assessment, and assurance evidence
3. define and develop key product and process measures and other performance indicators that can be used to validate a required level of assurance
4. collect and report measures that indicate the extent to which software and systems have achieved their required level of assurance
5. be able to perform assurance assessment for newly developed software and systems
6. be able to perform assurance assessment for acquired systems and services, including developing service level agreements and monitoring performance against such agreements

### Course: System Operational Assurance (7.1, 7.2, 7.3)

#### Catalog Description

This course teaches students how to establish procedures to assure that systems in operation continue to meet their security requirements and are able to respond to new threats. Topics include assurance policies and procedures; assurance training; technologies for monitoring and controlling systems; evaluating monitoring results; maintaining operational systems; evaluating malicious code; responding to adverse events; and taking actions necessary to maintain business survivability and continuity of operations.

#### Expected Outcomes

After completing this course, students will be able to

1. understand the role of business objectives and strategic planning in software and system assurance
2. create appropriate security policies and procedures for system operations
3. understand the type of training needed by users and administrative personnel in secure system operations
4. understand the capabilities and limitations of monitoring technologies for systems, services, and personnel
5. evaluate operational monitoring results for system and service functionality and security
6. maintain and evolve operational systems while preserving assured functionality and security
7. evaluate malicious content and apply appropriate countermeasures
8. plan for and execute effective responses to operational system accidents, failures, and intrusions
9. maintain business survivability and continuity of operations in adverse environments

### Course: System Security Assurance (5.1, 5.2, 5.3)

#### Catalog Description

This course teaches students how to incorporate effective security technologies and methods into new and existing systems. Students will learn how to think like an attacker in planning a variety of attacks, including password cracking, escalation of privileges, denial-of-service, and the creation, distribution, and insertion of viruses, worms, Trojans, spyware, logic bombs, and other malicious code. They will learn the most effective methods for preventing or defeating these attacks and analyzing the threats that they pose. Students will understand their ethical responsibilities and obligations when developing, acquiring, and operating software and systems.

#### Expected Outcomes

After completing this course, students will be able to

1. know the kinds of safety and security risks associated with critical infrastructure systems such as power, telecommunications, water, and air traffic control systems
2. understand the variety of methods by which attackers can damage software or data associated with software via weaknesses in the design or coding of the system
3. analyze threats to software
4. deploy appropriate countermeasures, such as layers, access controls, privileges, intrusion detection, encryption, and coding checklists
5. analyze threats to operational environments
6. design and plan for effective countermeasures such as access control, authentication, intrusion detection, encryption, and coding checklists
7. understand how physical security countermeasures such as gates, locks, guards, and background checks can address risks
8. understand how people who are knowledgeable about attack and prevention methods are obligated to use their abilities, both legally and ethically
9. understand the legal and ethical considerations involved in analyzing a variety of historical events and investigations

### Course: Assured Software Analytics (6.3)

#### Catalog Description

This course covers analysis methods, techniques, and tools to assure that newly developed and acquired software, systems, and services meet their functional and security requirements. Students will learn how to perform systems analysis, structural analysis, and functional analysis of software systems. They will also learn how to test for assurance and develop auditable assurance evidence.

#### Expected Outcomes

After completing this course, students will be able to

1. analyze system architectures, networks, and databases for assurance properties
2. restructure the logic of existing software to improve understandability and modifiability
3. reverse engineer existing software to determine functionality and security properties
4. understand the capabilities and limitations of methods, techniques, and tools for software analysis
5. evaluate testing methods, plans, and results for assuring software
6. develop auditable assurance evidence

### Course: Assured Software Development 1 (1.1, 1.2, 6.1, 6.2 [requirements])

#### Catalog Description

This course covers the fundamentals of incorporating assurance practices, methods, and technologies into software development and acquisition life-cycle processes and models. With this foundation, the course provides students with rigorous methods for eliciting software and system assurance requirements based on threat identification, characterization, and modeling; assurance risk management; and misuse/abuse cases. Students will also learn how to evaluate methods and environments for creating software and systems that meet their functionality and security requirements.

#### Expected Outcomes

After completing this course, students will be able to

1. understand life-cycle models and processes for newly developed software systems
2. understand life-cycle models and processes for the acquisition, supply, and service of a software system
3. use methods, techniques, and tools to assess the applicability of assurance processes and practices for typical life-cycle phases, such as requirements engineering, architecture and design, coding, testing, evolution, acquisition, and retirement
4. elicit and analyze requirements for assured software based on prior threat modeling, identification of attack patterns, and misuse/abuse cases
5. apply security requirements engineering methods in developing assurance requirements

### Course: Assured Software Development 2 (6.1, 6.2 [specification, design])

#### Catalog Description

This course covers rigorous methods for formal assurance specification and for architecting and designing software and systems to meet those specifications. Such methods include use of formal specification languages, applying security principles, architectural risk analysis, architectural vulnerability assessment, and technology-specific security guidelines.

#### Expected Outcomes

After completing this course students will be able to

1. use formal methods to specify and validate requirements for assured software
2. develop architectures that demonstrate that software and systems will satisfy their assurance requirements
3. design software and systems that fulfill architectural specifications for assurance
4. evaluate the capabilities and limitations of technical environments, languages, and tools when developing assured software
5. use assurance architecture and design methods, such as architectural risk analysis (including attack resistance, attack tolerance, and attack resilience), threat modeling, attack patterns, attack surface, design principles (such as least privilege and failing securely), and technology-specific guidelines
6. apply security technologies in developing architectures and designs, such as encryption, fault tolerance, intrusion detection, access controls, and authentication
7. understand quality attributes for software (including security) and how to specify them
8. understand design approaches for achieving quality attributes, including security, and tactics for achieving them

### Course: Assured Software Development 3 (6.2 [code, test, verification, validation])[26]

#### Catalog Description

This course covers rigorous methods, techniques, and tools for developing secure code. Such methods include code analysis for commonly known vulnerabilities, source code review using static analysis tools, and known language-specific practices for producing secure code.

This course also covers rigorous methods and tools for inspecting, testing, verifying, and validating software and systems to demonstrate that they meet functional and security requirements. Students will learn methods for verification and validation for security assurance and how security vulnerabilities can differ from programming errors. Team inspections and correctness verification methods will be covered. Testing techniques will include threat- and attack-based testing, functional testing, risk- and usage-based testing, stress testing, black- and white-box testing, and penetration testing.

#### Expected Outcomes

After completing this course, students will be able to

1. develop software that does not contain known vulnerabilities such as incorrect or incomplete input validation, poor or missing exception handling, buffer overflows, SQL injection, and race conditions
2. use methods, techniques, and tools that demonstrate that developed software meets its functionality and security requirements and implements its security architecture and design specifications
3. understand how to apply team inspections to validate functionality and security properties of software
4. understand methods for correctness verification of critical software components
5. understand how testing for security differs from traditional testing
6. test software to ensure that assurance requirements are met using a variety of methods, techniques, and tools
7. use threat models, attack patterns, and misuse/abuse cases during software and system testing
8. maintain software to continue to meet its functionality and security requirements

[26] This course may be taught as two separate courses depending on the level of content for secure coding and the extent to which students are expected to develop secure code.

## Course: Software Assurance Capstone Experience[27]

### Catalog Description

Students will work as part of a team to develop a secure software system for a customer. Deliverables include requirements specifications, preliminary and detailed designs, code and test verification, and validation results. The course culminates with a presentation of the software system to the customer, including a demonstration of its functional and security features.

### Expected Outcomes

After completing this course, students will be able to

1. establish and specify the required or desired level of assurance for a specific software system
2. evaluate the capabilities and limitations of technical environments, languages, and tools for assured software
3. identify, analyze, and perform software assurance practices that are relevant for the software to be developed
4. demonstrate compliance with laws, regulations, standards, and policies that apply to the software system
5. analyze the threats to which the software is most likely to be vulnerable in a specific operating environment and domain
6. develop requirements specification, and architecture and design specifications, that satisfy the required/desired level of assurance for a specific software system
7. apply methods, techniques, and tools to construct software modules that meet the functionality and security requirements and implement their security architecture and design specifications
8. apply testing and review methods, develop plans, and analyze results that demonstrate that a software system satisfies its functionality and security requirements
9. plan for and ensure that the software responds effectively to operational software accidents, failures, and intrusions

### Special Topics in the Capstone Experience

*Network-Based Assurance*

Students will focus on issues that occur in web applications, mobile computing, cloud computing, and systems of systems.

*Software Safety and Reliability*

Students will focus on assuring safety and reliability of software systems; hazard/risk analyses; Fault Tree Analysis (FTA), Failure Mode and Effect Analysis (FMEA), and Event Tree Analysis (ETA); design and implementation diversity; fault tolerance; and so on.

[27] This course may be offered over two terms.

*Performance Analysis of Computer Systems*

Students will focus on using quantitative methods to assess the system, statistical analysis, analysis of variance (ANOVA), experiment design, measurements, simulation, and analytical models.

*Fault Tolerant Systems*

Students will focus on sources of faults and failures, redundant designs, and multi-version programming.

*Model-Based Verification*

Students will focus on formal and semi-formal representation of systems and their verification.

## MSwA Courses Added to MSwE Program (Seven Courses)

### Course: Assurance Management (1.2, 2.1, 2.2, 2.3, 4.1, 4.2, 4.3)

#### Catalog Description

This course covers the fundamentals of assurance management, including risk analysis and tradeoff assessment of security measures, business cases for software assurance, standards and regulations related to software assurance, and the planning and control of projects involving software assurance.

#### Expected Outcomes

After completing this course, students will be able to

1. make a business case for assurance
2. assess the use of software assurance processes and practices
3. understand how to add assurance considerations and practices as part of normal project management activities
4. identify, analyze, and select assurance practices that are relevant for a specific software development project
5. understand laws, regulations, standards, and policies that are relevant to assurance
6. understand basic risk management concepts
7. identify, analyze, plan for, mitigate, and monitor assurance risks
8. identify risks arising from vulnerabilities and threats
9. determine assurance processes and practices that mitigate risks

### Course: System Operational Assurance (3.1, 3.2, 3.3, 6.4, 7.1, 7.2, 7.3)

#### Catalog Description

This course teaches students how to analyze and validate the effectiveness of assurance operations, create auditable evidence of security measures, monitor and assess system operational security, and respond to new threats.

#### Expected Outcomes

After completing this course, students will be able to

1. establish and specify the required or desired level of assurance for a specific software system
2. understand appropriate methods for assessment of software assurance
3. use appropriate product and process measures by software development life-cycle phase
4. define additional performance indicators for software systems and processes
5. collect and report measures that demonstrate effective software assurance
6. understand the role of business objectives and strategic planning in system assurance
7. create appropriate security policies and procedures for system operations
8. understand the type of training needed by users and administrative personnel in secure system operations
9. understand the capabilities and limitations of monitoring technologies for systems, services, and personnel
10. evaluate operational monitoring results for system and service functionality and security
11. maintain and evolve operational systems while preserving assured functionality and security
12. evaluate malicious content and apply appropriate countermeasures
13. plan for and execute effective responses to operational system accidents, failures, and intrusions
14. maintain business survivability and continuity of operations in adverse environments

### Course: System Security Assurance (5.1, 5.2, 5.3) [This course description is the same as the MSwA Stand-Alone Program.]

### Course: Assured Software Analytics (6.3) [This course description is the same as the MSwA Stand-Alone Program.]

### Course: Assured Software Development 1 (1.1, 6.1, 6.2 [requirements, specification, design])

#### Catalog Description

This course covers the fundamentals of incorporating assurance practices, methods, and technologies into software development life-cycle processes and models. With this foundation, the course provides students with rigorous methods for eliciting and specifying software and system assurance requirements and for developing architectures and designs that meet assurance requirements. Students will also learn how to evaluate technologies for creating software and systems that meet their functionality and security requirements.

#### Expected Outcomes

After completing this course, students will be able to

1. understand life-cycle models and processes for newly developed software and systems
2. use methods, techniques, and tools to assess the applicability of assurance processes and practices for typical life-cycle phases, such as requirements engineering, architecture and design, coding, testing, evolution, acquisition, and retirement
3. elicit, analyze, specify, and validate requirements for assured software
4. architect and design assured software
5. evaluate the capabilities and limitations of technical environments, languages, and tools when developing assurance requirements, architectures, and designs
6. understand requirements and design approaches for achieving quality attributes, including security and tactics for achieving them

### Course: Assured Software Development 2 (6.2 [code, test, verification, validation])

#### Catalog Description

This course covers rigorous methods for implementing (coding), testing, verifying, and validating software and systems to demonstrate that they meet functional and security requirements.

#### Expected Outcomes

After completing this course, students will be able to

1. develop software that does not contain known vulnerabilities
2. use methods, techniques, and tools that demonstrate that developed software meets its functional and security requirements and implements its security architecture and design specifications
3. test, verify, and validate software to ensure that assurance requirements are met using a variety of methods, techniques, and tools
4. maintain software to continue to meet its functionality and security requirements

### Course: Software Assurance Capstone Experience [This course description is the same as the MSwA Stand-Alone Program.]

#### Special Topics in the Capstone Experience [Same as MSwA Stand-Alone]

The special topics are the same as the MSwA Stand-Alone Program *plus* acquisition. Acquisition is added as a special topic here because it does not receive the same level of attention in the MSwE program as in the MSwA Stand-Alone Program.

# Glossary

### acquisition
Process of obtaining a system, software product, or software service. Software products may include commercial, off-the-shelf (COTS) products; modified, off-the-shelf (MOTS) products; open source products; or fully developed products.

The above definition was derived from these references:

[IEEE-CS 2008]
*ISO/IEC 12207, IEEE Std 12207-2008, Systems and Software Engineering - Software Life Cycle Processes*

[IEEE-CS 1998]
*IEEE Std 1062, IEEE Recommended Practice for Software Acquisition*

### correct functionality
Software assurance seeks to provide a level of confidence that software functions in the intended manner as defined by requirements and specifications. Software should establish a secure computing environment and provide required functionality that is free from errors and known vulnerabilities. Software evolution should maintain these properties.

### development models
Include incremental, spiral, evolutionary, and agile methods.

### diverse systems
Include systems-of-systems, network systems, embedded systems, critical infrastructure systems, service-oriented systems, industrial networks, supervisory control and data acquisition systems (SCADA), distributed control systems (DCS), COTS, legacy systems, and open source software. Awareness and understanding as applied to diverse systems may include

- analysis of system boundaries, interfaces with service providers, and service level agreement to assure required performance
- evaluation of network system design—throughput, load balancing, backup and recovery, and operational monitoring to assure required availability
- analysis of system-of-systems integration and interoperability to assure preservation of security properties and required functional behavior
- assurance of computational and information asset preservation and continuity of operations through backup and switchover methods
- assurance of security properties—including authentication, authorization, integrity, confidentiality, non-repudiation, and privacy across a variety of system architectures, configurations, and providers

### life-cycle processes
Include new system development, legacy system evolution, and acquisition, both for systems through supply chains, open source, and COTS, and for services through external providers. Also includes process models such as CMMI.

### security controls
The management, operational, and technical controls (that is, safeguards or countermeasures) prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information.

### security objective
Confidentiality, integrity, or availability.

### security properties
Include authentication, authorization, confidentiality, integrity, non-repudiation, and privacy.

### software analytics
Include reverse-engineering technologies that transform arbitrary control logic into structured form and function abstraction to recover designs and specifications from implementations.

### software analytics
Specialized technologies and processes are necessary to analyze and assure functional and security properties of software. Analysis subject matter extends across the life cycle and includes specification, design, code, inspection, and test artifacts. Analytic methods include reverse engineering to transform arbitrary control logic into structured form for improved understanding and function abstraction to recover designs and specifications from implementations.

### software assurance
Application of technologies and processes to achieve a required level of confidence[28] 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.

This definition has been expanded from the original definition offered by The Committee on National Security Systems [CNSS 2009].

### Software quality
Capability of a software product to satisfy stated and implied needs when used under specified conditions [ISO 2009].

### software security
Engineering software so that it is as vulnerability- and defect-free as possible and continues to function correctly in spite of attack or misuse.

---

[28] In the CNSS definition, the use of the word “confidence” implies that there is a basis for the belief that software systems and services function in the intended manner.

### system defenses
Include filtering, monitoring, and control at network, system, and application levels and specific technologies including encryption and multi-layering.

### system vulnerabilities
Include system architecture, design, implementation, operational, and user characteristics that enable attack strategies.

### threat environments
Include attack sources, motivations, technologies, methods, targets, and consequences.

# Acronyms

**ACM**
Association for Computing Machinery

**ANOVA**
analysis of variance

**ASEE**
American Society for Engineering Education

**Bloom Cognitive Levels**
K—knowledge
C—comprehension
AP—application
AN—analysis

**BoK**
body of knowledge

**BSI**
Build Security In

**CAPEC**
*Common Attack Pattern Enumeration and Classification*

**CIO**
chief information officer

**CISO**
chief information security officer

**CMMI**
Capability Maturity Model Integration

**CMU**
Carnegie Mellon University

**CNSS**
Committee of National Security Systems

**COTS**
commercial, off-the-shelf

**CWE**
common weakness enumeration

**DCS**
distributed control systems

**DHS**
Department of Homeland Security

**DoD**
Department of Defense

**DOS**
disk operating system

**ECTS**
European Credit Transfer and Accumulation System

**ERAU**
Embry-Riddle Aeronautical University

**ETA**
event tree analysis

**FISMA**
Federal Information Security Management Act

**FMEA**
Failure Mode and Effect Analysis

**FTA**
fault tree analysis

**GOTS**
government, off-the-shelf

**GSwE2009**
Graduate Software Engineering Curriculum

**HIPAA**
Health Insurance Portability and Accountability Act

**HR**
human resources

**IA**
information assurance

**IAB**
industry advisory board

**IEEE**
Institute of Electrical and Electronics Engineers

**IEEE-CS**
Institute of Electrical and Electronics Engineers Computer Society

**iSSEc**
Integrated Software & Systems Engineering Curriculum

**ISO**
International Organization for Standardization

**IT**
information technology

**KA**
knowledge area

**KU**
knowledge unit

**MBA**
Master of Business Administration

**MOTS**
modified-off-the-shelf

**MSE**
Master of Software Engineering

**MSSE**
Master of Science in Software Engineering

**MSwA**
Master of Software Assurance

**MSwA Core BoK**
Master of Software Assurance Core Body of Knowledge

**MSwA2010**
Master of Software Assurance Curriculum

**MSwA2010 BoK**
Master of Software Assurance Curriculum Body of Knowledge

**MSwE**
Master of Software Engineering

**NISPOM**
National Industrial Security Program Operating Manual

**NIST**
National Institute of Standards and Technology

**NCSD**
National Cyber Security Division

**OPM**
U.S. Office of Personnel Management

**OS**
operating system

**OWASP**
*Open Web Application Security Project*

**PM**
project manager

**PPS**
Partnership for Public Service

**RFP**
request for proposal

**QA**
quality assurance

**SA**
software assurance

**SCADA**
Supervisory Control and Data Acquisition Systems

**SDLC**
software development life cycle

**SE**
software engineering

**SEEPP**
Software Engineering Ethics and Professional Practices

**SEI**
Software Engineering Institute

**SIA**
survivability and information assurance

**SOA**
service-oriented architecture

**SPOC**
single point of contact

**SQL**
Structured Query Language

**SwA**
software assurance

**SwA BoK**
Software Assurance Body of Knowledge

**SwACBK**
Software Assurance Curriculum Body of Knowledge

**SWEBOK**
Software Engineering Curriculum Body of Knowledge

# References

*URLs are valid as of the publication date of this report.*


### [ACM 2008]
The Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). "Computer Science Curriculum 2008: An Interim Revision of CS 2001." *Computing Curriculum Series*. http://www.acm.org//education/curricula/ComputerScience2008.pdf (2008).

### [ACM 2009]
The Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP). http://www.acm.org/about/se-code (2009).

### [Allen 2008]
Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Professional, 2008.

### [Ardis 1989]
Ardis, Mark A. & Ford, Gary. *SEI Report on Graduate Software Engineering Education (1989)* (CMU/SEI-89-TR-021). Software Engineering Institute, Carnegie Mellon University, 1989. http://www.sei.cmu.edu/library/abstracts/reports/89tr021.cfm

### [Bloom 1956]
Bloom, B. S., ed. *Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I, Cognitive Domain*. Longmans, 1956.

### [CERT 2007]
CERT. *Survivability and Information Assurance Curriculum.* Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/sia/ (2007).

### [CNSS 2009]
Committee on National Security Systems (CNSS). *Instruction No. 4009, National Information Assurance Glossary*. Revised June 2009.

### [DHS 2010a]
Department of Homeland Security (DHS) Software Assurance (SwA). *Build Security In*. https://buildsecurityin.us-cert.gov/daisy/adm-bsi/home.html (2010).

### [DHS 2010b]
Department of Homeland Security (DHS) Software Assurance (SwA) Workforce Education and Training Working Group. *Software Assurance CBK/Principles Organization.* https://buildsecurityin.us-cert.gov/bsi/dhs/927-BSI.html (2010).

**[Drew 2009]**
Drew, Christopher. "Wanted: 'Cyber Ninjas.'" *New York Times*, December 29, 2009. http://www.nytimes.com/2010/01/03/education/edlife/03cybersecurity.html?emc=eta1(Accessed January 2010).

**[Ellison 2010]**
Ellison, Robert J.; Goodenough, John B.; Weinstock, Charles B.; & Woody, Carol. *Evaluating and Mitigating Software Supply Chain Security Risks* (CMU/SEI-2010-TN-016). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn016.cfm

**[ERAU 2010]**
Embry-Riddle Aeronautical University. *Master of Software Engineering Program*. http://www.erau.edu/db/degrees/ma-softwareeng.html (2010).

**[Ford 1991]**
Ford, Gary. *1991 SEI Report on Graduate Software Engineering Education* (CMU/SEI-91-TR-002). Software Engineering Institute, Carnegie Mellon University, 1991. http://www.sei.cmu.edu/library/abstracts/reports/91tr002.cfm

**[Huitt 2006]**
Huitt, W. "The Cognitive System." *Educational Psychology Interactive*. Valdosta State University, 2006. http://chiron.valdosta.edu/whuitt/col/cogsys/cogsys.html (Accessed May 22, 2008).

**[IEEE-CS 2004a]**
IEEE Computer Society (IEEE-CS) & the Association for Computing Machinery (ACM). "Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering." *Computing Curriculum Series.* http://www.acm.org/education/education/curric_vols/CE-Final-Report.pdf (2004).

**[IEEE-CS 2004b]**
IEEE Computer Society (IEEE-CS) & the Association for Computing Machinery (ACM). "Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." *Computing Curriculum Series.* http://sites.computer.org/ccse/SE2004Volume.pdf (2004).

**[IEEE-CS 2004c]**
IEEE Computer Society (IEEE-CS). *Software Engineering Body of Knowledge (SWEBOK).* http://www.computer.org/portal/web/swebok (2004).

**[iSSEc 2009]**
Integrated Software & Systems Engineering Curriculum (iSSEc) Project. *Graduate Software Engineering 2009 (GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering, Version 1.0*. Stevens Institute of Technology, 2009.

**[ISO 2009]**
International Organization for Standardization. *ISO/IEC/IEEE 24765 - Systems and Software Engineering – Vocabulary*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=50518 (2009).

**[LSEC 2009]**
Leaders in Security. “Building In ... Information Security, Privacy And Assurance.” Paper presented at the Knowledge Transfer Network Paris Information Security Workshop. Paris, France, March 30, 2009.

**[Mead 2010]**
Mead, Nancy R.; Hilburn, Thomas B.; & Linger, Rick. *Software Assurance Curriculum Project Volume II: Undergraduate Course Outlines* (CMU/SEI-2010-TR-019, ESC-TR-2010-019). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr019.cfm

**[OPM 2010]**
U.S. Office of Personnel Management. *Federal Cyber Service: Scholarship for Service.* https://www.sfs.opm.gov/ (2010).

**[PPS 2009]**
Partnership for Public Service & Booz Allen Hamilton. *Cyber IN-Security: Strengthening the Federal Cybersecurity Workforce*. Partnership for Public Service. http://ourpublicservice.org/OPS/publications/viewcontentdetails.php?id=135 (July 2009).

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| (Leave Blank) | August 2010 | Final |

| 4. TITLE AND SUBTITLE | 5. FUNDING NUMBERS |
|---|---|
| Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum | FA8721-05-C-0003 |

6. AUTHOR(S)

Nancy R. Mead, Julia H. Allen, Mark Ardis, Thomas B. Hilburn, Andrew J. Kornecki, Richard Linger, James McDonald

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| Software Engineering Institute<br>Carnegie Mellon University<br>Pittsburgh, PA 15213 | CMU/SEI-2010-TR-005 |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |
|---|---|
| HQ ESC/XPK<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 | ESC-TR-2010-005 |

11. SUPPLEMENTARY NOTES

| 12A DISTRIBUTION/AVAILABILITY STATEMENT | 12B DISTRIBUTION CODE |
|---|---|
| Unclassified/Unlimited, DTIC, NTIS | |

13. ABSTRACT (MAXIMUM 200 WORDS)

Modern society depends on software systems of ever-increasing scope and complexity in virtually every sphere of human activity, including business, finance, energy, transportation, education, communication, government, and defense. Because the consequences of failure can be severe, dependable functionality and security are essential. As a result, software assurance is emerging as an important discipline for the development, acquisition, and operation of software systems and services that provide requisite levels of dependability and security.

This report is the first volume in the Software Assurance Curriculum Project sponsored by the U.S. Department of Homeland Security. This report presents a body of knowledge from which to create a Master of Software Assurance degree program, as both a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. The report details the process used to create the curriculum and presents the body of knowledge, curriculum architecture, student prerequisites, and expected student outcomes. It also outlines an implementation plan for faculty and other professionals who are responsible for designing, developing, and maintaining graduate software engineering programs that have a focus on software assurance knowledge and practices. The second volume, *Undergraduate Course Outlines* (CMU/SEI-2010-TR-019), presents seven course outlines that could be used in an undergraduate curriculum specialization for software assurance.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| software assurance, software assurance education, software engineering education, software security education | 154 |

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |