Software Engineering Institute

Software Assurance Curriculum Project Volume IV: Community College Education

Nancy R. Mead, Software Engineering Institute
Elizabeth K. Hawthorne, Union County College
Mark Ardis, Stevens Institute of Technology

September 2011

TECHNICAL REPORT
CMU/SEI-2011-TR-017
ESC-TR-2011-017

CERT® Program

http://www.sei.cmu.edu

Carnegie Mellon

Table of Contents

List of Tables

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.
- The DHS NCSD Workforce Education & Training Working Group provided valuable review comments on the draft document.

The following individuals contributed to the earlier Association for Computing Machinery (ACM) Community College Course Outlines:

- Robert D. Campbell, City University of New York Graduate Center
- Karl J. Klee, Alfred State College of Technology
- Anita M. Wright, Camden County College

We also appreciate the thoughtful peer review provided by

- Julia H. Allen, Software Engineering Institute
- Thomas B. Hilburn, Embry-Riddle Aeronautical University
- Andrew J. Kornecki, Embry-Riddle Aeronautical University

We acknowledge the contributions by Jennifer Kent as report editor and Shannon Haas in helping to assemble the report.

Abstract

The fourth volume in the Software Assurance Curriculum Project led by a team at the Software Engineering Institute, this report focuses on community college courses for software assurance. The report includes a review of related curricula, outcomes and body of knowledge, expected background of target audiences, and outlines of six courses. The courses are intended to provide students with fundamental skills for continuing with graduate-level education or to provide supplementary education for students with prior undergraduate technical degrees who wish to become more specialized in software assurance.

Previous volumes of this project are *Volume I: Master of Software Assurance Reference Curriculum*, *Volume II: Undergraduate Course Outlines*, and *Volume III: Software Assurance Course Syllabi*.

1 Introduction

Nearly every facet of modern society depends heavily on highly complex software systems. The business, energy, transportation, education, communication, government, and defense communities rely on software to function, and software is an intrinsic part of our personal lives. Software assurance is an important discipline to ensure that software systems and services function dependably and are secure.

Recognizing the importance of the software assurance discipline for protecting national infrastructures and systems, the U.S. Department of Homeland Security (DHS) has recognized the growing need for skilled practitioners in this area. At the direction of the DHS, the Software Engineering Institute (SEI) at Carnegie Mellon University developed the Software Assurance Curriculum Project. Volume I is the *Master of Software Assurance Reference Curriculum* (MSwA2010) [Mead 2010a], Volume II is the *Undergraduate Course Outlines* [Mead 2010b], and Volume III is the *Master of Software Assurance Course Syllabi* [Mead 2011].

This report, Volume IV, focuses on community college courses for software assurance. In addition to the earlier volumes of the Software Assurance Curriculum Project, the Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges (CCECC) Computer Science Curriculum was a primary resource in the development of this report [ACM 2009]. The foundation of the Software Assurance Curriculum Project includes the SEI's work on the DHS Build Security In website [DHS 2011a] and work by DHS on the Software Assurance Curriculum Body of Knowledge (SwACBK) [DHS 2011b].

The courses outlined in this document are intended to provide students with fundamental skills for continuing with undergraduate-level education or supplementary education for students with prior undergraduate technical degrees who wish to become more specialized in software assurance.

Definition of Software Assurance

In developing the *Master of Software Assurance Reference Curriculum*, the authors started with a clear definition of "software assurance." They used as the foundation the definition from the Committee on National Security Systems as follows [CNSS 2009]:

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

For purposes of the *Master of Software Assurance Reference Curriculum* report, the authors expanded the CNSS definition as follows [Mead 2010a]:

> *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*

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.

intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

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

2 Review of Related Curricula

Based on our team's expertise, we did a brief literature search and considered the following existing curricula as possible sources for this report:

- CyberWatch Information Assurance Curriculum
- Software Engineering 2004 (SE 2004) Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering [IEEE-CS 2004b]
- ACM Committee for Computing Education in Community Colleges (CCECC) Computer Science Curriculum [ACM CCECC 2009a]
- Information Assurance (IA) Curricula Guidelines (ITiCSE Working Group Guidelines) [Cooper 2010]
- Survivability and Information Assurance (SIA) Curriculum [CERT 2007]
- Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) Computing Curricula 2001 (CC 2001) [IEEE-CS 2001]
- IEEE-CS and ACM Computing Curricula 2008 (CS 2008) [IEEE-CS 2008]
- SEI SwA Undergraduate Course Outlines [Mead 2010b]

Brief synopses of the publicly available materials follow, and additional details are in Appendix A.

CyberWatch Information Assurance Curriculum

CyberWatch is an Advanced Technological Education (ATE) Center, headquartered at Prince George's Community College and funded by a grant from the National Science Foundation (NSF). CyberWatch has four model information assurance programs available:

- Associate of Applied Science (A.A.S.) in Information Assurance
- Associate of Science (A.S.) in Information Assurance
- Certificate in Information Assurance
- Certificate in Information Assurance Management

CyberWatch has six of the eight model courses it developed currently available for download. In addition, CyberWatch has virtual lab facilities to assist its member institutions in delivering the IA courses and assists with curriculum development that emphasizes creating associate's degree and certificate programs from a core set of technical and industry certification courses.

Software Engineering 2004 (SE 2004) Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering

These curriculum guidelines provide guidance on what should be included in an undergraduate software engineering education. The recommendations include the skills and knowledge that every software engineering graduates should know as well as ways to teach those skills and knowledge.

ACM Committee for Computing Education in Community Colleges (CCECC) Computer Science Curriculum

The foundation for the computer science associate-degree transfer program is the three-course sequence of Computer Science I, Computer Science II, and Computer Science III. Students can take additional computing courses based on factors like transfer requirements, institutional specializations, and student interests.

Information Assurance (IA) Curricula Guidelines (ITiCSE WG Guidelines)

The ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE) Information Assurance (IA) Curriculum Guidelines Working Group examined existing IA academic program curricula and governmental and industry IA standards and guidelines. The working group used this information as the foundation for proposing a body of knowledge for IA.

Survivability and Information Assurance (SIA) Curriculum

The CERT® Program at the SEI published the SIA Curriculum in 2006. The three-course curriculum offers a problem-solving methodology built on key SIA principles that are independent of specific technologies. The major topic areas that correspond to each course are

1. Principles of Survivability and Information Assurance: This course details the ten principles on which the entire SIA curriculum is based.
2. Information Assurance Networking Fundamentals: This course applies the ten principles to the concepts and an implementation of Transmission Control Protocol (TCP)/Internet Protocol (TCP/IP) networking.
3. Sustaining, Improving, and Building Survivable Functional Units (SFUs)

IEEE-CS and ACM Computing Curricula 2001 (CC 2001)

The Computing Curricula 2001 project is a joint effort of IEEE-CS and ACM. CC 2001 makes recommendations for undergraduate programs in computer science based on the report's computer science body of knowledge, the computer science undergraduate core material, learning objectives, curriculum models, and course descriptions.

IEEE-CS and ACM Computing Curricula 2008 (CS 2008)

After the Computing Curricula was published in 2001, which focused on recommendations for computer science undergraduate programs, IEEE-CS and ACM published additional volumes of recommendations for computer engineering, information systems, information technology, and software engineering. CS 2008 is an interim review report of the original CC 2001 that also considers the additional volumes and input from those in industry and academia. CS 2008

® CERT is a registered trademark owned by Carnegie Mellon University.

includes an update of the CC 2001 Body of Knowledge, with additional commentary and recommendations in the text.

Software Assurance Undergraduate Course Outlines

These course outlines are the second volume in the SEI's Software Assurance Curriculum Project. The seven course outlines are intended to provide students with an undergraduate curriculum specialization in software assurance. The goal is for students to be equipped with fundamental skills for either entering the field directly or continuing with graduate-level education.

Conclusions

After reviewing the above curricula, we were able to sharpen our focus. We recognized that existing curricular models designated as either information assurance or information security were not the primary topics of our effort, and so curricula in those areas were less helpful to us. Rather, we were focused on software assurance, while recognizing that in a two-year program, students would typically not be able to complete more than three to four related courses. To this end, the sources that we considered most useful included CS 2008, CC 2001, the ACM CCECC Curriculum, and the ITiCSE Working Group Guidelines. SE 2004, the Undergraduate Course Outlines, and the SIA Curriculum were considered partially useful. We did not evaluate certificate programs, training programs leading to certificates, or individual community college curricula.

3 Outcomes and Body of Knowledge

To identify the desired outcomes for software assurance at the community college level, we started with the Master of Software Assurance (MSwA) Body of Knowledge (BoK) taken from the *Master of Software Assurance Reference Curriculum* [Mead 2010a]. The MSwA BoK is organized by outcome and indicates the level of knowledge that a student should obtain for each topic area. The knowledge levels are represented by the following Bloom's cognitive levels, which are described in detail in Appendix B:

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

We started with the existing MSwA knowledge levels and refined them for bachelor of science (BS) and associate of science (AS) degrees. The results of this exercise follow in Table 1.

Table 1: Software Assurance BOK Across Curriculum Levels

BoK Topics	MSwA	BS	AS
1. Assurance Across Life Cycles			
1.1. Software Life-Cycle Processes			
New development	C	C	K
Processes associated with the full development of a software product.			
Integration, assembly, and deployment	C	C	K
Processes concerned with the final phases of the development of a new or modified software product			
Operation and evolution	C	K	NA
Processes that guide the operation of the a software product and its change over time			
Acquisition, supply, and service	C	K	NA
Processes that support acquisition, supply, or service of a software product			
1.2. Software Assurance Processes and Practices			
Process/practice assessment	AP	K	NA
Methods, procedures, and tools used to assess assurance processes and practices			
Software assurance integration into SDLC phases	AP	C	K
The integration of assurance practices into typical life-cycle phases (e.g., requirements engineering, architecture and design, coding, test, evolution, and acquisition)			
2. Risk Management			
2.1. Risk Management Concepts			
Types and classification	C	K	K
Describe the different classes of risks (e.g., business, project, technical)			
Probability, impact, severity	C	K	NA
Describe basic elements of risk analysis			
Models, processes, metrics	C	K	NA

BoK Topics	MSwA	BS	AS
Understand various models, processes, and metrics used in risk management			
2.2.Risk Management Process			
Identification	AP	C	K
Identify and classify risks associated with a project			
Analysis	AP	C	NA
Analyze the likelihood, impact, and severity of each identified risk			
Planning	AP	K	NA
Develop a risk management plan covering risk avoidance and mitigation			
Monitoring and management	AP	K	NA
Assess and monitor risk occurrence and manage risk mitigation			
2.3. Software Assurance Risk Management			
Vulnerability and threat identification	AP	C	K
Application of risk analysis techniques to vulnerability and threat risks			
Analysis of software assurance risks	AP	C	K
Analyze risks for both new and existing systems			
Software assurance risk mitigation	AP	K	NA
Plan for and mitigate software assurance risks			
Assessment of software assurance processes and practices	AP	K	NA
As part of risk avoidance and mitigation, assess the identification and use of appropriate software assurance processes and practices			
3. Assurance Assessment			
3.1. Assurance Assessment Concepts			
Baseline level of assurance; allowable tolerances, if quantitative	AP	K	NA
Establish and specify the required/desired level of assurance for a specific software application, set of applications, or a software-reliant system (and tolerance for same)			
Assessment methods	C	C	K
Understand how various methods (such as validation of security requirements, risk analysis, threat analysis, vulnerability assessments/scans, and assurance cases) can be used to determine if the software/system being assessed is sufficiently secure within tolerances			
3.2. Measurement for Assessing Assurance			
Product and process measures by life-cycle phase	AP	K	NA
Define and develop key product and process measurements that can be used to validate the required level of software assurance appropriate to a given life-cycle phase			
Other performance indicators that test for the baseline, by life-cycle phase	AP	K	NA
Define and develop additional performance indicators that can be used to validate the required level of software assurance appropriate to a given life-cycle phase			
Measurement processes and frameworks	C	NA	NA
Understand a range of software assurance measurement processes and frameworks and how these might be used to accomplish 1.2.			
Business survivability and operational continuity	AP	NA	NA
Define and develop 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			
Comparison of selected measurements to the established baseline	AP	NA	NA

BoK Topics	MSwA	BS	AS
Analyze key product and process measures and performance indicators to determine if they are within tolerance when compared to the defined baseline			
Identification of out-of-tolerance variances	AP	NA	NA
Be able to identify measures that are out of tolerance when compared to the defined baselines and be able to develop actions to reduce the variance			
4. Assurance Management			
4.1. Making the Business Case for Assurance			
Valuation and cost/benefit models, cost and loss avoidance, return on investment	AP	K	NA
Apply financially-based approaches, methods, models, and tools to develop and communicate compelling cost/benefit arguments in support of deploying software assurance practices			
Risk analysis	C	K	K
Understand how risk analysis can be used to develop cost/benefit arguments in support of deploying software assurance practices			
Compliance justification	C	K	K
Understand how compliance with laws, regulations, standards, and policies can be used to develop cost/benefit arguments in support of deploying software assurance practices			
Business impact/needs analysis	C	K	NA
Understand 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			
Project management across the life cycle	C	K	NA
Understand how to lead software and system assurance efforts as an extension of normal software development (and acquisition) project management skills			
Integration of other knowledge units	AN	C	K
Be able to identify, analyze, and select software assurance practices from any knowledge units that are relevant for a specific software development or acquisition project			
4.3. Compliance Considerations for Assurance			
Laws and regulations	C	NA	K
Understand the extent to which selected laws and regulations are relevant for a specific software development or acquisition project, and how compliance might be demonstrated			
Standards	C	K	K
Understand the extent to which selected standards are relevant for a specific software development or acquisition project, and how compliance might be demonstrated			
Policies	C	NA	NA
Understand 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			
5.1. For Newly Developed and Acquired Software for Diverse Applications			
Security/safety aspect of computer intensive critical infrastructure systems such as power, telecommunication, water, and air traffic control	K	K	K

BoK Topics	MSwA	BS	AS
Know the kinds of safety and security risks associated with critical infrastructure systems such as power, telecommunications, water, and air traffic control systems			
Potential attack methods	C	K	K
Understand the variety of methods by which attackers can damage software or data associated with that software via weaknesses in the design or coding of the system			
Analysis of threats to software	AP	K	NA
Analyze the threats to which software is most likely to be vulnerable in specific operating environments and domains			
Methods of defense	AP	K	K
Be familiar with appropriate countermeasures such as layers, access controls, privileges, intrusion detection, encryption, and coding checklists			
5.2 For Diverse Operational (Existing) Systems			
Historic and potential operational attack methods	C	K	NA
Understand and be able to duplicate the attacks that have been used to interfere with an application's or system's operations			
Analysis of threats to operational environments	AN	C	NA
Analyze the threats to which software is most likely to be vulnerable in specific operating environments and domains			
Designing and planning for access control, privileges, and authentication	AP	C	NA
Design and plan for access control and authentication			
Security methods for physical and personnel environments	AP	C	NA
Understand how gates, locks, guards, and background checks can address risks			
5.3 Ethics and Integrity in Creation, Acquisition, and Operation of Software Systems			
Overview of ethics, code of ethics, and legal constraints	C	K	K
Understand how people who are knowledgeable about attack and prevention methods are obligated to use their abilities, both legally and ethically			
Computer attack case studies	C	NA	NA
Understand the legal and ethical considerations involved in analyzing a variety of historical events and investigations			
6. **System Functionality Assurance**			
6.1. Assurance Technology			
Technology evaluation	AN	NA	NA
Evaluating capabilities and limitations of technical environments, languages, and tools with respect to creating assured software functionality and security			
Technology improvement	AP	NA	NA
Recommending improvements in technology as necessary within project constraints			
6.2. Assured Software Development			
Development methods	AP	AP/C	AP/C/K
Rigorous methods for system requirements, specification, design, implementation, verification, and testing to develop assured software			
Quality attributes	C	C	K
Software quality properties and how to achieve them			
Maintenance methods	AP	C	NA
Assurance aspects of software maintenance and evolution			
6.3. Assured Software Analytics			

BoK Topics	MSwA	BS	AS
Systems analysis	AP	K	NA
Analyzing system architectures, networks, and databases for assurance properties			
Structural analysis	AP	K	NA
Structuring the logic of existing software to improve understandability and modifiability			
Functional analysis	AP	K	NA
Reverse engineering of existing software to determine functionality and security properties			
Analysis methods and tools	C	NA	NA
Capabilities and limitations of methods and tools for software analysis			
Testing for assurance	AN	K	NA
Evaluating testing methods, plans, and results for assuring software			
Assurance evidence	AP	NA	NA
Developing auditable assurance evidence			
6.4. Assurance in Acquisition			
Assurance of acquired software	AP	K	NA
Assuring software acquired through supply chains, vendors, and open sources, including developing requirements and assuring delivered functionality and security			
Assurance of software services	AP	K	NA
Developing service-level agreements for functionality and security with service providers and monitoring compliance			
7. System Operational Assurance			
7.1. Operational Procedures			
Business objectives	C	K	NA
Role of business objectives and strategic planning in system assurance			
Assurance procedures	AP	K	NA
Creation of security policies and procedures for system operations			
Assurance training	C	NA	NA
Evaluation of training for users and administrative personnel in secure system operations			
7.2. Operational Monitoring			
Monitoring technology	C	NA	NA
Capabilities and limitations of monitoring technologies, and installation and configuration or acquisition of monitors and controls for systems, services, and personnel			
Operational evaluation	AP	K	NA
Evaluation of operational monitoring results with respect to system and service functionality and security			
Operational maintenance	AP	K	NA
Maintenance and evolution of operational systems while preserving assured functionality and security			
Malware analysis	AP	K	NA
Evaluation of malicious content and application of countermeasures			
7.3. System Control			
Responses to adverse events	AN	K	NA
Planning for and executing effective responses to operational system accidents, failures, and intrusions			
Business survivability	AP	K	NA

BoK Topics	MSwA	BS	AS
Maintenance of business survivability and continuity of operations in adverse environments			

Although this is a valid mapping, the BoK topics are not distributed evenly across the MSwA curriculum. For example, item 6.2 in the BoK covers many topic areas, and this is where many of the community college topics appear. Some of the other items in the BoK do not apply to community college courses at all. Therefore, this exercise was not that helpful for the level of detail needed in the community college courses. We therefore decided to build the course outlines and associated outcomes from the bottom up. We used the proposed IA body of knowledge in the ITiCSE workshop report [Cooper 2010] and the courses from the ACM CCECC computer science transfer curriculum [ACM 2009] as a base.

4 Target Audience and Expected Background

We did a brief survey of the profiles of community college students. Summaries of the relevant results appear below, with extracted material in Appendix C.

According to the American Association for Community Colleges, more than half of U.S. undergraduate students have attended community college. Community colleges provide access to postsecondary education that minority, low income, and first-generation college students may not otherwise have. Community colleges prepare students for transfer to four-year institutions, help working adults prepare for new careers, and offer noncredit programs that offer a range of knowledge and skills, like learning a new language.

According to the Community College Research Center located at Teachers College, Columbia University, most community college students are older than 25, though many students are also recent high school graduates who want to cost effectively start their college education.

According to the presentation CTE Dual Enrollment: Preparing Students for College and Careers [Hughes 2011], community college computing[2] students include

- recent high school graduates interested in a career as computer programmers or game developers who want to earn an associate's degree before transferring to a four-year institution
- students who have completed an undergraduate degree in a field other than computing and want to learn about computing so they can either use the skills in their current career or enter a computing career
- students who have completed an undergraduate degree in computing and want to update their knowledge and skills
- students interested in either earning a technical certificate, indicating that they have completed a specific set of courses in a specialty area, or who are just looking to learn a skill and not interested in a certificate
- students in a computing career who need a credential, like CISSP, to further their career

Although the students attending community colleges are quite diverse, the courses outlined in this report are intended to provide all these types of students with fundamental skills for continuing with undergraduate-level education or supplementary education for students with prior undergraduate technical degrees who wish to become more specialized in software assurance.

[2] Community college computing here refers to computer science, information technology, and other broad computing topics.

5 Overview of Courses

Courses at the community college level are typically three to four credits each. The Computer Science I–II–III course sequence, typical at community colleges as well as smaller four-year colleges, is the equivalent of the Computer Science I–II course sequence at other four-year colleges and universities. See guidance for introductory courses described in Appendix A, Computing Curricula 2001 section. In addition, a student might form a specialty by taking two to three elective courses. Associate computing degrees are typically in computer science (CS), information technology (IT), or information systems (IS). The specialties (such as SwA in our case) may be defined formally by the individual colleges and appear in the catalog, but they do not appear on the diploma. We concluded that an appropriate selection of courses for an SwA specialty could include Computer Science I, II, and III and more specialized courses such as Introduction to Computer Security, Secure Coding, and Introduction to Assured Software Engineering. These are not intended to be an exhaustive list of possible courses but rather a set of courses that could reasonably be taken by students wishing to pursue further education in software assurance.

The six courses that we describe in this report appear in two different formats. Since Computer Science I, II, and III include updates to existing course descriptions from the ACM CCECC we decided to retain that original format, which is closer to a syllabus with learning outcomes for assessment than an outline of course topics. The other three courses, which are more specialized, appear in the outline format that was used in Volume II of our software assurance education report series [Mead 2010b]. Currently, we do not have enough information or actual experience to describe them in more detail as community college courses. Brief descriptions of all six courses follow, and the syllabi and outlines are in Sections 6 through 11.

Computer Science I: This course is the first in a three-course sequence that provides students with a foundation in computer science. Students develop fundamental programming skills using a language that supports an object-oriented approach, secure coding awareness, human-computer interactions, and social responsibility.

Computer Science II: This course is the second in a three-course sequence that provides students with a foundation in computer science. Students develop intermediate programming skills using a language that supports an object-oriented approach, with an emphasis on algorithms, software development, secure coding techniques, and ethical conduct.

Computer Science III: This course is the third in a three-course sequence that provides students with a foundation in computer science. Students develop advanced programming skills using a language that supports an object-oriented approach, with an emphasis on data structures, algorithmic analysis, software engineering principles, software assurance checklists, and professionalism.

Introduction to Computer Security: This course provides an overview of the fundamentals of computer security. Topics include security standards, policies, and best practices; principles, mechanisms, and implementation of computer security and data protection; security policy, encryption, and authentication; access control and integrity models and mechanisms; network security; secure systems; programming and vulnerabilities analysis; principles of ethical and professional behavior; regulatory compliance and legal issues; information assurance; risk management and threat assessment; business continuity and disaster recovery planning; and security across the life cycle.

Secure Coding: This course covers security vulnerabilities of programming in weakly typed languages like C and in more modern languages like Java. Common weaknesses exploited by attackers are discussed, as well as mitigation strategies to prevent those weaknesses. Students practice programming and analysis of software systems through testing and static analysis. Topics covered include methods for preventing unauthorized access or manipulation of data, input validation and user authentication, memory management issues related to overflow and corruption, misuse of strings and pointers, and inter-process communication vulnerabilities.

Introduction to Assured Software Engineering: This course covers the basic principles and concepts of assured software engineering; system requirements; secure programming in the large; modeling and testing; object-oriented analysis and design using the unified modeling language (UML); design patterns; frameworks and application programming interfaces (APIs); client-server architecture; user interface technology; and the analysis, design and programming of extensible software systems.

6 Computer Science I

Course Description

This course is the first in a three-course sequence that provides students with a foundation in computer science. Students develop fundamental programming skills using a language that supports an object-oriented approach, incorporating secure coding, human-computer interactions, and social responsibility.

Prerequisites

- Computer fluency (no previous programming or computer science experience expected)
- Precalculus-ready (that is, proficiency sufficient to enter college-level precalculus course)
- English Composition I-ready (that is, proficiency sufficient to enter college-level English I course)

Co-Requisite

Discrete Structures

Syllabus

Course Minimum Contact Hours: 42 (recommended hours per topic identified below)

Table 2: Syllabus for Computer Science I Course

Topic	Bloom's Level
Secure coding (2 hours): data protection techniques of input validation, data encapsulation, information hiding and integrity, and strict data typing	A
Fundamental programming constructs (11 hours): basic syntax and semantics of a higher-level language; variables (scope and lifetime), types, expressions, and assignment; self-documentation; standard and file input/output; conditional and iterative control structures; structured decomposition; pseudo-random number generator	A
Fundamental algorithms and problem-solving (6 hours): problem-solving strategies; the role of algorithms in the problem-solving process; implementation strategies for algorithms; debugging strategies; the concept and properties of algorithms	A
Fundamental data structures (6 hours): primitive types, arrays, records, strings, references	A
Object-oriented principles (6 hours): abstraction, objects, classes, methods, parameter passing, encapsulation, inheritance, polymorphism	A
Program development (3 hours): program development phases, with emphasis on design, implementation, and testing and debugging strategies	A
Software tools and integrated development environment (IDE) (2 hours): compiling, interpreting, linking, executing, testing, and debugging	A
Programming languages (1 hour): comparison of object-oriented, procedural, functional programming	C
Human-computer interaction (1 hour): sound design concepts and fundamental graphical interface design; standard API graphics	C
Machine-level representation of data (1 hour): overview of the storage of instructions, numbers, and characters in a Von Neumann machine	C

Topic	Bloom's Level
Ethical conduct (1 hour): codes of ethics and responsible conduct; intellectual property, copyright, and plagiarism; "Ten Commandments for Computer Ethics"	C
Overview of operating systems (1 hour): role and purpose of operating systems; simple file management	C
Historical context of computing (1 hour): history of computing ideas, computing, and programming	K

Sources

ACM Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Taylor, B. & Shiva Azadegan, "Moving beyond security tracks: Integrating security in CS0 and CS1." [Taylor 2008]

Additional Items

Course Student Learning Outcomes

Upon successful completion of this course, the student will be able to

- choose professional behavior in response to ethical issues inherent in computing
- produce algorithms for solving simple problems and trace the execution of computer programs
- compare and contrast the primitive data types of a programming language; describe how each is stored in memory; and identify the criteria for selection
- apply the program development and testing process to problems that are solved using fundamental programming constructs and predefined data structures
- explain the need for secure coding techniques as applied to object-oriented programming solutions
- decompose a program into subtasks and use parameter passing to exchange information between the subparts
- differentiate between object-oriented, structured, and functional programming methodologies
- describe the language translation phases of compiling, interpreting, linking, and executing and differentiate the error conditions associated with each phase

Course Assessment Features

The following table was taken from the ACM CCECC Assessment Rubric for Computer Science I [ACM CCECC 2009a].

Table 3: Course Assessment Features for Computer Science I Course

Course Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
Apply secure coding techniques to object-oriented programming solutions.	Describes secure coding techniques of an object-oriented program, such as public versus private members, data integrity, and data typing.	Applies secure coding techniques to an object-oriented program.	Devises a fully secure object-oriented program.
Apply the program development process to problems that are solved using fundamental programming constructs and predefined data structures.	Summarizes the phases of the program development cycle.	With guidance during the design phase, produces working code and performs some testing.	Develops a working program solution by implementing design, coding, and testing that includes error checking.
Choose professional behavior in response to ethical issues inherent in computing.	Explains the concepts of intellectual property, plagiarism, and software piracy.	Chooses to respond professionally to ethical issues in computing, such as intellectual property, plagiarism, and software piracy.	Values and respects intellectual property and chooses to act professionally.
Compare and contrast the primitive data types of a programming language; describe how each is stored in memory; and identify the criteria for selection.	Names the built-in data types of the programming language.	Differentiates among the built-in data types and explains when it is appropriate to choose one over another.	Consistently produces programming solutions with the correct data types implemented.
Decompose a program into subtasks and use parameter passing to exchange information between the subparts.	With guidance, translates a problem into a programming solution with subtasks.	With guidance for program analysis and design, decomposes a problem into program components that share data.	Independently analyzes a problem, formulates a design strategy, and decomposes a problem into program components that share data.
Describe the language translation phases of compiling, interpreting, linking, and executing and differentiate the error conditions associated with each phase.	Defines the programming language terms of compiling, interpreting, linking, executing, and error conditions.	Describes the programming language translation phases of compiling, interpreting, linking, and executing.	Compares the programming language translation phases of compiling, interpreting, linking, and executing and distinguishes the error conditions associated with each.
Differentiate between the object-oriented, structured, and functional programming methodologies.	Recognizes the differences a□d similarities of the object-oriented, structured, and functional programming methodologies.	Differentiates between the object-oriented, structured, and functional programming methodologies.	Compares and contrasts the three prominent methodologies of object-oriented, structured, and functional programming.
Produce algorithms for solving simple problems and trace the execution of computer programs.	Defines the steps necessary to solve a programming problem.	Produces a working programming solution for a given algorithm.	Develops a generic solution for an algorithm that can be used to solve a range of related problems.

7 Computer Science II

Course Description

This course is the second in a three-course sequence that provides students with a foundation in computer science. Students develop intermediate programming skills using a language that supports an object-oriented approach, with an emphasis on algorithms, software development, software assurance and ethical conduct.

Prerequisites

- Computer Science I
- Discrete Structures

Co-Requisite

Calculus I

Syllabus

Course Minimum Contact Hours: 42 (recommended hours per topic identified below)

Table 4: Syllabus for Computer Science II Course

Topic	Bloom's Level
Secure coding (3 hours): buffer overflows; memory leaks; malicious code; unauthorized and back-door access; security-aware exception handling	A
Software development (4 hours): software life cycle; test case design; software tools; debuggers and simulators; characteristics of maintainable software; program code verification and data validation; software inspection	A
Object-oriented programming (7 hours): encapsulation and information hiding; inheritance; class hierarchies; polymorphism; abstract and interface classes	A
Object-oriented design and modeling (5 hours): class constructors and destructors; abstract data types (ADTs); reusable software components; APIs; modeling tools; class diagrams	A
Intermediate programming constructs (3 hours): cohesion and decoupling; assertions, including pre/post conditions and loop invariants; software reuse; self-documentation	A
Intermediate computing algorithms (5 hours): searching; sorting; recursive algorithms; complexity of algorithms	A
Intermediate data structures (7 hours): built-in; programmer-created; dynamic	A
Event-driven programming (4 hours): graphics API; event creation; event-handling methods; exception handling	A
Human-computer interaction (2 hours): sound design concepts; interfaces between people and technology	C
Simple database integration (1 hour): database I/O; embedded SQL queries; SQL injection	C
Societal and professional issues (1 hour): computing and the internet; social impact of computing; privacy	C

Sources

ACM Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Stoneburner Gary; Hayden,Clark; & Feringa, Alexis. "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A (NIST Special Publication 800-27 Rev A)." [Stoneburner 2004]

Additional Items

Course Student Learning Outcomes

Upon successful completion of this course, the student will be able to

- discuss significant trends and societal impacts related to computing, software, and the internet
- construct object-oriented programming solutions for reuse, using ADTs that incorporate encapsulation, data abstraction, and information hiding
- construct multiple-file or multiple-module programming solutions that use class hierarchies, inheritance, and polymorphism to reuse existing design and code
- design and develop secure programs that mitigate the most common security vulnerabilities
- verify program correctness through the development of sound test plans and the implementation of comprehensive test cases and software inspections
- create programming solutions that use data structures and existing libraries

Course Assessment Features

The following table was taken from the ACM CCECC Assessment Rubric for Computer Science II [ACM CCECC 2009b].

Table 5: Course Assessment Features for Computer Science II Course

Course Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
Analyze the execution of searching and sorting algorithms.	Describes the execution trace of one searching algorithm and one sorting algorithm.	Analyzes the execution of various searching and sorting algorithms.	Evaluates the execution of various searching and sorting algorithms including a recursive solution.
Construct multiple-file or multiple-module programming solutions that use class hierarchies, inheritance, and polymorphism to reuse existing design and code.	Describes when inheritance and the use of class hierarchies is an appropriate design strategy.	With guidance, produces a programming solution using inheritance and polymorphism.	Designs and constructs a programming solution using the features of inheritance and polymorphism appropriately.

Course Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
Construct object-oriented programming solutions for reuse, using ADTs that incorporate encapsulation, data abstraction, and information hiding.	Summarizes the concepts of encapsulation, data abstraction, and information hiding and explains how they apply to object-oriented programming.	Organizes programming solutions that include encapsulation, information hiding, and data abstraction.	Constructs reusable software components that incorporate encapsulation, data abstraction, and information hiding.
Create programming solutions that use data structures and existing libraries.	Produces programming solutions that use existing library code.	Organizes programming solutions that incorporate appropriate data structures and pre-existing code.	Designs and develops programming solutions that use data structures, pre-existing libraries, and individual library code.
Design and develop secure and fault-tolerant programs that mitigate potential security vulnerabilities.	Summarizes important characteristics of software assurance, such as the elimination of buffer overflows, memory leaks, and back-door access.	Produces a program using the foundations of software assurance to mitigate potential security vulnerabilities.	Designs and develops a secure programming solution using principles of software assurance.
Discuss significant trends and societal impacts related to computing, software, and the internet.	Explains how databases and the internet can impact privacy and property rights.	Discusses the potential uses and abuses of data and the consequences of the loss of privacy.	Practices ethical behavior when addressing property rights and privacy issues.
Produce graphical user interfaces that incorporate simple color models and handle events.	Differentiates between good and bad design concepts for human-computer interfaces.	Produces programming code of a graphical user interface that utilizes a simple color model effectively, and efficiently handles events triggered by user interaction.	Develops programming code for a graphical user interface that incorporates the concepts of good human-computer interaction (HCI) design.
Verify program correctness through the development of sound test plans and the implementation of comprehensive test cases.	Produces test plans for object-oriented programming solutions that consider code coverage.	Analyzes a program and devises a test plan that examines code coverage and develops test cases for data coverage.	Constructs a test driver for code coverage and creates a formal test plan choosing comprehensive test cases for data coverage.

8 Computer Science III

Course Description

This course is the third in a three-course sequence that provides students with a foundation in computer science. Students develop advanced programming skills using a language that supports an object-oriented approach, with an emphasis on data structures, algorithmic analysis, software engineering principles, software assurance, and professionalism.

Prerequisites

- Computer Science II
- Calculus I

Syllabus

Course Minimum Contact Hours: 42 (recommended hours per topic identified below)

Table 6: Syllabus for Computer Science III Course

Topic	Bloom's Level
Software assurance (3 hours): conformance with assurance coding standards and practices, trustworthiness, and predictable execution testing; quality reviews; engineering and security tradeoffs; risks and liabilities of computer-based systems; fault prevention in software life-cycle stages; intentional and unintentional software security vulnerabilities.	A
Formal computing algorithms (8 hours): efficiency of various sorting and searching algorithms; hashing; collision-avoidance strategies; binary search trees; depth- and breadth-first traversals; shortest-path algorithms; minimum spanning tree; transitive closure; topological sort	A
Canonical data structures (7 hours): stacks; queues; linked lists; hash tables; trees; graphs	A
Recursion (7 hours): recursive mathematical functions; divide-and-conquer, first-and-rest, and last-and-rest strategies; backtracking; recursion with linked lists; trees; graphs	A
Software reuse (3 hours): design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators	A
Human-computer interaction (2 hours): universal principles; human-centered considerations; usability testing and verification; design tradeoffs; secure user interfaces	C
Software engineering (4 hours): standard approaches and implementation tools for analysis and design; measurement and metrics; software life-cycle stages, processes, and documentation; software process maturity scale	C
Algorithmic strategies (2 hours): brute-force; greedy; branch-and-bound; heuristics; pattern matching; string/text	C
Basic algorithmic analysis (3 hours): asymptotic analysis of upper and average complexity bounds; best, average, and worst case behaviors; Big-O and little o notations; standard complexity classes; empirical measurements of performance; time and space tradeoffs; recurrence relations	C
Concurrency (2 hours): threads; scheduling, synchronization and timing; multi-threaded programs; race conditions	C
Professionalism (1 hour): standards of professional behavior; professional computing societies and publications; professional responsibilities and liabilities; ACM Code of Conduct; career paths in computing	C

Sources

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Stoneburner Gary; Hayden,Clark; & Feringa, Alexis. “Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A (NIST Special Publication 800-27 Rev A).” [Stoneburner 2004]

Association for Computing Machinery (ACM), Inc. *ACM Code of Ethics and Professional Conduct*. [ACM 2011]

Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS) Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. [ACM 1999]

Additional Items

Course Student Learning Outcomes

Upon successful completion of this course, the student will be able to

- practice the tenets of ethical and professional behavior promoted by professional societies and accept the professional responsibilities and liabilities associated with software development
- use standard analysis and design techniques to produce a team-developed, medium-sized software application that is fully implemented and formally tested for the elimination of common software security vulnerabilities
- compare and contrast a range of searching and sorting algorithms and analyze time and space efficiencies
- assess the appropriateness of using recursion to solve a given problem
- design and construct programming solutions using a variety of recursive techniques
- analyze the efficiency of recursive algorithms
- design and develop reusable software using appropriate data structures and templates
- create effective, efficient, and secure software, reflecting standard principles of software engineering and software assurance

Course Assessment Features

The following table was taken from the ACM CCECC Assessment Rubric for Computer Science III [ACM CCECC 2009c].

Table 7: Course Assessment Features for Computer Science III Course

Course Learning Outcome	**Approaches Goal**	**Meets Goal**	**Surpasses Goal**
Analyze the efficiency of recursive algorithms.	With guidance, interprets a recursive method.	Analyzes a recursive method and correctly predicts its output.	Evaluates recursive algorithms in terms of efficiency and time and space tradeoffs.
Assess the appropriateness of using recursion to solve a given problem.	Explains the utility of recursion to solve certain problems.	Compares and contrasts the tradeoffs in terms of recursive and non-recursive solutions.	Justifies when to choose a recursive solution over a non-recursive solution (and vice versa) in terms of efficiency, Big-O, and comprehensibility.
Compare and contrast a range of searching and sorting algorithms and analyze time and space efficiencies.	Uses various searching and sorting algorithms, and investigates time and space tradeoffs.	Compares and contrasts a range of searching and sorting algorithms for time and space efficiencies.	Critiques searching and sorting algorithms, including recursive solutions, for various algorithmic efficiencies and evaluates them in terms of Big-O.
Create effective, efficient and secure software, reflecting standard principles of software engineering and software assurance.	Ranks the risks and liabilities of a computer-based solution using standard software assurance and engineering principles.	Creates an effective, efficient, and secure solution, utilizing principles of software assurance and software engineering.	Judges the security of a software solution.
Design and construct programming solutions using a variety of recursive techniques.	Converts a simple recursive algorithm into a working recursive method.	With guidance, develops recursive programming solutions for applications that use data structures such as trees and lists.	Independently designs and develops recursive programming solutions for applications that use backtracking and data structures such as trees and lists.
Design and develop reusable software using appropriate data structures and templates.	Differentiates among the classic data structures and selects a suitable data structure for use in an application.	With guidance, designs and develops applications using appropriate data structures for a given problem.	Independently designs and develops applications using appropriate data structures and incorporates reusable software components in the solution.
Practice the tenets of ethics and professional behavior promoted by computing societies; accept the professional responsibilities and liabilities associated with software development.	Studies the tenets of ethics and professional behavior promoted by international computing societies, such as ACM and IEEE-CS.	Practices the tenets of ethics and professional behavior promoted by international computing societies and recognizes the liabilities associated with software development.	Displays ethical and professional behavior associated with the responsibilities of software development.
Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software application that is fully implemented and formally tested.	As part of a team, produces an executable, medium-sized software application that meets some program requirements and includes design	As part of a team, produces a working, medium-sized software application on time that meets many program requirements including design and some test plan documentation.	As part of a team, successfully develops a medium-sized, secure software application on time that meets all program requirements including design and formal test plan documentation.

Course Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
	documentation and some evidence of testing.		

9 Introduction to Computer Security

Course Description

This course provides an overview of the fundamentals of computer security. Topics include security standards, policies, and best practices; principles, mechanisms, and implementation of computer security and data protection; security policy, encryption, and authentication; access control and integrity models and mechanisms; network security; secure systems; programming and vulnerabilities analysis; principles of ethical and professional behavior; regulatory compliance and legal issues; information assurance; risk management and threat assessment; business continuity and disaster recovery planning; and security across the life cycle (requirements, architecture and design, construction, testing, operation, maintenance, acquisition, and services).

Prerequisites

Computer Science I

Outline

Table 8: Outline for Introduction to Computer Security Course

Topic	Bloom's Level
Security goals and fundamentals: confidentiality, integrity, availability, reliability, etc.	K
Secure systems: types, models, design, changes to non-secure systems; comparative analysis	C
Access controls: controlling access to resources, access matrix model, access control lists and capability lists; mandatory controls, originator controls	C
Networks and security: internet security architecture, internet protocols, implementation considerations; firewalls	C
Integrity: cryptographic checksums, malicious logic, viruses, Trojan horses; defenses, prevention	K
Cryptography fundamentals: classical, public key; implementation problems	K
Authentication: passwords	C
Attacks: software attacks (malicious code, buffer overflows, social engineering, injection attacks, and related defense tools); network attacks (denial of service, flooding, sniffing and traffic redirection, defense tools and strategies); website attacks (cross-site scripting)	K
Management: planning for security; introduction to risk assessment and management; business cases; regulatory compliance and legal issues; Federal Information Security Management Act; and business continuity/disaster planning	K
Security standards in government and industry: NIST 800-39 (risk management), NIST 800-53 (security controls), ISO 27001, and ISO 27002; sample corporate and institutional security policies	K
Security issues in requirements, architecture, design, implementation, testing, operation, maintenance, acquisition, and services	K
Ethics and professionalism as related to computer security	K

Sources

Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). "Computer Science Curriculum 2008: An Interim Revision of CS 2001." *Computing Curriculum Series.* [ACM 2008]

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

Bishop, Matt. *Computer Security: Art and Science.* [Bishop 2002]

Redwine, Samuel T., Jr. *Secure Software Engineering Education.* This is a description of the first three years of the master's program in secure software engineering at James Madison University, including a somewhat detailed description of a one-semester course in secure software engineering. [Redwine 2010]

Stallings, W. *Network Security Essentials.* [Stallings 2007]

Wright, Marie & Kakalik, John. *Information Security: Contemporary Cases.* [Wright 2006]

Additional Items

Course Delivery Features

In addition to conventional lecture and discussion methods, the following techniques are appropriate for course delivery:

- This course provides an excellent opportunity to use case studies. *Information Security: Contemporary Cases* by Wright and Kakalik provides a source for reading, study, and case-study exercises.
- A number of hands-on individual and team projects could be assigned; for example
  - an exercise or discussion of quality attributes related to security
  - an exercise involving the review of program code to identify security problems
  - comprehension of the security shortcomings of an existing software artifact or a computing system (standalone application, network, operating system, website)
  - presentation on a current security technology or issue

Course Assessment Features

Most of the course topics are listed at the K (Knowledge) Bloom's level, which means that students need a basic familiarity with the topics. They must be able to discuss and describe at a level that shows appreciation of computer security issues. Assessment of the results of the activities and exercises discussed in the "Course Delivery Features" section is a good way of judging achievement at the specified Bloom's level.

10 Secure Coding

Course Description

This course covers security vulnerabilities of programming in weakly typed languages like C and in more modern languages like Java. Common weaknesses exploited by attackers are discussed, as well as mitigation strategies to prevent those weaknesses. Students practice programming and analysis of software systems through testing and static analysis.

Prerequisites and Co-Requisites

4. Computer Science I as a prerequisite (with experience programming in C or C++), or
5. Computer Science II as a co-requisite (with experience coding in Java), otherwise
6. Computer Science II as a prerequisite

Syllabus

Table 9: Syllabus for Secure Coding Course

Topic	Bloom's Level
Overview of security vulnerabilities and risks in software: Common Weakness Enumeration (CWE), Open Web Application Security Project (OWASP) Top 10	C
Data protection: methods for preventing unauthorized access or manipulation of data	AP
Input validation and user authentication	AP
Memory management: buffer overflow, memory corruption, and privilege violations	AP
Integer overflow and misuse of strings and pointers	AP
Communication vulnerabilities: concurrency, secure inter-process communication and authorization, authentication and networking protocols	AP
Unit testing for security vulnerabilities: fuzzing, abuse cases	AP
Code review: formal inspections and static analysis	AP
Vulnerabilities in modern languages: insecurities in Java and hypertext preprocessor (PHP)	C
Standard risk mitigation strategies and resources: coding standards, enterprise security API (ESAPI)	C
Professional development: OWASP, certification	C

Sources

Cooper, Stephen; Nickell, Christine; Pérez, Lance C.; Oldfield, Brenda; Brynielsson, Joel; Gencer Gökce, Asim; Hawthorne, Elizabeth K.; Klee, Karl J.; Lawrence, Andrea; & Wetzel, Susanne. *Towards Information Assurance (IA) Curricular Guidelines* (ITiCSE 2010 Working Group Report). [Cooper 2010]

Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development*. [Merkow 2010]

Seacord, Robert C. *Secure Coding in C and C++*. [Seacord 2005]

Additional Items

Course Delivery Features

In addition to conventional lecture and discussion methods, a number of hands-on individual and team projects are appropriate for course delivery; for example

- individual programming projects (including coding using a coding standard, preparation of a unit test plan, and test results)
- individual or team review/inspections
- individual or team testing and analysis of artifacts produced by other students or available online

Course Assessment Features

Many of the course topics are listed at the AP (Application) Bloom's level, which means that students must be able to use information, methods, concepts, and theories to solve problems that require the skills or knowledge taught in the course. For topics labeled C (Comprehension), students must be able to discuss, describe, and interpret the topics. Assessment of the results of the activities and exercises discussed in the "Course Delivery Features" section is a good way of judging achievement at the specified Bloom's level.

11 Introduction to Assured Software Engineering

Course Description

This course covers the basic principles and concepts of assured software engineering; system requirements; secure programming in the large; modeling and testing; object-oriented analysis and design using the UML; design patterns; frameworks and APIs; client-server architecture; user interface technology; and the analysis, design, and programming of extensible software systems.

Prerequisite

Computer Science II

Co-Requisite

Computer Science III

Syllabus

This syllabus is largely based on the ACM CCECC course descriptions modified and abstracted to the outline level [ACM CCECC 2009d, ACM CCECC 2009e].

Table 10: Syllabus for Introduction to Assured Software Engineering Course

Topic	Bloom's Level
Introduction to software project management: project planning, estimation, configuration management, risk management; and software security process models: Building Security In Maturity Model (BSIMM), OWASP Software Assurance Maturity Model (SAMM), Microsoft Software Development Lifecycle (SDL)	C
Role of assured software engineering: software engineering for assurance and its place as an engineering discipline	C
Requirements analysis: requirements analysis for functional and quality requirements	AP
Introduction to software architecture: introduction to software architecture, including architectural patterns (pipe & filter, MVC), client-server computing	C
Use and misuse cases: use cases, misuse cases, and user-centered design	C
Design patterns: abstraction-occurrence, composite, player-role, singleton, observer, delegation, facade, adapter, etc.	C
UML: review of object-oriented principles, UML class diagrams, and object-oriented analysis	AP
Domain modeling: examples of building class diagrams to model various domains	C
Reusable technologies: review of reusable technologies as a basis for software engineering, risks associated with reuse (e.g. Ariane)	C
Software behavior: representing software behavior: sequence diagrams, state machines, activity diagrams, correctness under all conditions of use	AP
Verification and validation: inspections and reviews, integration, system, and acceptance testing	AP

Sources

ACM Committee for Computing Education in Community Colleges (CCECC). *Program Details: Introduction to Software Engineering*. [ACM CCECC 2009d]

ACM Committee for Computing Education in Community Colleges (CCECC). *Course Details: Introduction to Software Engineering*. [ACM CCECC 2009e]

IEEE Computer Society (IEEE-CS) & Association for Computing Machinery (ACM). "Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." *Computing Curriculum Series*. [IEEE-CS 2004b]

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

Redwine, Samuel T., Jr. *Secure Software Engineering Education.* This is a description of the first three years of the master's program in secure software engineering at James Madison University, including a somewhat detailed description of a one-semester course in secure software engineering [Redwine 2010].

Additional Items

Course Delivery Features

Sample labs and assignments include the following:

- evaluating the assurance and performance of various simple software designs
- adding features, including assurance features, to an existing system
- testing a system to verify conformance to test cases including assurance
- building a graphical user interface (GUI) for an application
- numerous exercises building models in UML, particularly class diagrams and state machines
- developing and presenting a simple set of assured software requirements (to be done as a team) for some innovative client server application of very small size
- implementing the above, using reusable technology to the greatest extent possible, while understanding the risks associated with reuse

In addition to conventional lecture and discussion methods, the following techniques are appropriate for course delivery:

- This course is a good starting point for exposing students to moderately sized existing systems. With such systems, students can learn and practice the essential skills of reading and understanding code written by others. Students should write secure code in the context of a particular domain, for example, the biological, physical, mathematical, or chemical sciences or even wider spectra such as game programming, business applications, and graphics and animation.
- We suggest that a core subset of UML be taught rather than trying to cover all features.
- It may be challenging for instructors to convey the nature of software engineering to students; however, this challenge may be addressed through strategies such as field trips to businesses and industries that utilize large software systems, guest lectures by developers and users of large software systems, and discussions about embedded systems in everyday

life including automated teller machines (ATMs), wireless devices, cell phones, various mobile devices, and computer games and their associated risks and vulnerabilities.

Course Assessment Features

The depth of coverage of the course topics varies, as do the associated Bloom's levels. In many areas, students need to be able to discuss and describe the topics, but in other areas they must be able to apply the techniques learned in the course to actual software projects. In general they must be able to discuss, describe, and apply the techniques at a level that shows appreciation of assured software engineering. Assessment of the results of the activities and exercises discussed in the "Course Delivery Features" section is a good way of judging achievement at the specified Bloom's level.

Table 11: Typical Introduction to Assured Software Engineering Course Sequence Option 1

Term 1	Term 2	Term 3	Term 4
CS I	CS II	CS III	Secure Coding
Discrete Structures	Calculus I	Assured Software Engineering	
	Introduction to Computer Security		

Table 12: Typical Introduction to Assured Software Engineering Course Sequence Option 2

Term 1	Term 2	Term 3	Term 4
CS I	CS II	CS III	Secure Coding
Discrete Structures	Calculus I	Introduction to Computer Security	Assured Software Engineering

12 Resources

Computer Science I

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Computer Science II

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Stoneburner Gary; Hayden,Clark; & Feringa, Alexis. "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A (NIST Special Publication 800-27 Rev A)." [Stoneburner 2004]

Computer Science III

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges. *ACM Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. [ACM 2009]

Stoneburner Gary; Hayden,Clark; & Feringa, Alexis. "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A (NIST Special Publication 800-27 Rev A)." [Stoneburner 2004]

Association for Computing Machinery (ACM), Inc. *ACM Code of Ethics and Professional Conduct*. [ACM 2011]

Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS) Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. [ACM 1999]

Introduction to Computer Security

Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). "Computer Science Curriculum 2008: An Interim Revision of CS 2001." *Computing Curriculum Series.* [ACM 2008]

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

Bishop, Matt. *Computer Security: Art and Science.* [Bishop 2002]

Redwine, Samuel T., Jr. *Secure Software Engineering Education.* This is a description of the first three years of the master's program in secure software engineering at James Madison University, including a somewhat detailed description of a one-semester course in secure software engineering. [Redwine 2010]

Stallings, W. *Network Security Essentials.* [Stallings 2007]

Wright, Marie & Kakalik, John. *Information Security: Contemporary Cases.* [Wright 2006]

Secure Coding

Cooper, Stephen; Nickell, Christine; Pérez, Lance C.; Oldfield, Brenda; Brynielsson, Joel; Gencer Gökce, Asim; Hawthorne, Elizabeth K.; Klee, Karl J.; Lawrence, Andrea; & Wetzel, Susanne. *Towards Information Assurance (IA) Curricular Guidelines* (ACM ITiCSE 2010 Working Group Report). [Cooper 2010]

Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development*. [Merkow 2010]

Seacord, Robert C. *Secure Coding in C and C++*. [Seacord 2005]

Introduction to Assured Software Engineering

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges (CCECC). *Program Details: Introduction to Software Engineering*. [ACM CCECC 2009d]

Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges (CCECC). *Course Details: Introduction to Software Engineering*. [ACM CCECC 2009e]

IEEE Computer Society (IEEE-CS) & Association for Computing Machinery (ACM). "Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." *Computing Curriculum Series*. [IEEE-CS 2004b]

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

Redwine, Samuel T., Jr. *Secure Software Engineering Education.* This is a description of the first three years of the master's program in secure software engineering at James Madison University, including a somewhat detailed description of a one-semester course in secure software engineering [Redwine 2010].

Websites of Interest

Build Security In

Sponsored by the DHS National Cyber Security Division (NCSD), this website provides practices, tools, guidelines, rules, principles, and other resources that software developers, architects, and security practitioners can use to build security into software in every phase of its development.

https://buildsecurityin.us-cert.gov/bsi/home.html

CERT Podcasts

The CERT podcast series provides both general principles and specific starting points for business leaders who want to launch an enterprise-wide security effort or make sure their existing security program is as good as it can be.

http://www.cert.org/podcast/

National Software Assurance Repository

The National Software Assurance Repository (NSAR) is a Department of Defense/National Security Agency funded knowledge base of commonly accepted current practices, principles, methodologies, and tools for software assurance. The NSAR incorporates as many life-cycle methodologies and tools for assuring software as could be identified in the literature. It also itemizes all related supporting principles and concepts to ensure the security of internally and externally developed and sustained software.

http://cybersecurity.udmercy.edu/manage/search.php

SEI Virtual Training Environment

The SEI Virtual Training Environment (VTE) amplifies the training and best practices the SEI has developed and delivered in the classroom. Because of the rich media instruction and hands-on training labs, VTE allows users to access high-quality training material anywhere in the world, with only a Web browser and an Internet connection.

https://www.vte.cert.org/vteWeb/

Software Assurance Community Resource Information Clearinghouse

Also sponsored by the DHS NCSD, this website provides additional resources on many software assurance topics including workforce education and training, processes and practices, technology and tools, acquisition and outsourcing, measurement, establishing a business case, and malware.

https://buildsecurityin.us-cert.gov/swa/

Appendix A: Related Curricula

CyberWatch Curriculum

The following information is from http://www.cyberwatchcenter.org [CyberWatch 2011].

> Four CyberWatch model Information Assurance programs have been developed and are available from CyberWatch:
>
> - A.A.S. in Information Assurance
> - A.S. in Information Assurance
> - Certificate in Information Assurance
> - Certificate in Information Assurance Management
>
> CyberWatch has developed eight model courses, six of which are currently available for download. We have an active program to make all of these courses also available in an online format, with an expected completion date of Fall 2010:
>
> - CW 110 Ethics in the Information Age
> - CW 130 Microcomputer Operating Systems
> - CW 160 Security+
> - CW 225 Hardening the Infrastructure
> - CW 230 Microsoft Windows Server 2003
> - CW 235 Network Defense and Countermeasures
> - CW xxx Computer Forensics I (not yet available for download)
> - CW xxx Disaster Recovery and Risk Management (not yet available for download)
>
> CyberWatch maintains virtual lab facilities to assist member institutions in the delivery of IA courses. These include the CyberWatch Virtual Lab, the Digital Forensics Lab, and the CyberWatch Underground Tunnel System.
>
> CyberWatch provides assistance to member institutions for curriculum development and for mapping of courses to the Committee on National Security Systems (CNSS) 4011 and 4013 national IA training standards. Curriculum development emphasizes building associate's degree and certificate programs from a set of core technical courses that, in addition to meeting 4011 and/or 4013 standards, help prepare students for several industry certifications including:
>
> - CompTIA's Network+ and Security+
> - Cisco Certified Network Associate (CCNA)
> - Microsoft Certified Professional (MCP)
> - Security Certified Network Professional (SCNP)

Software Engineering 2004 (SE 2004) Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering

The following information is from http://sites.computer.org/ccse [IEEE-CS 2004b]

> The primary purpose of this volume is to provide guidance to academic institutions and accreditation agencies about what should constitute an undergraduate software engineering education. These recommendations have been developed by a broad, internationally based group of volunteer participants. This group has taken into account much of the work that has been done in software engineering education over the last quarter of a century. Software engineering curriculum recommendations are of particular relevance, since there is currently a surge in the creation of software engineering degree programs and accreditation processes for such programs have been established in a number of countries.
>
> The recommendations included in this volume are based on a high-level set of characteristics of software engineering graduates presented in Chapter 3. Flowing from these outcomes are the two main contributions of this document:
>
> - SEEK: Software Engineering Education Knowledge - what every SE graduate must know
> - Curriculum: ways that this knowledge and the skills fundamental to software engineering can be taught in various contexts

ACM Committee for Computing Education in Community Colleges (CCECC) Computer Science Curriculum

The following information is from
http://www.acmccecc.org/pgm_inventory/programdetail.aspx?pID=38 [ACM CCECC 2009a].

> The foundation for the Computer Science associate-degree transfer program is the three-course computing sequence CS I - CS II - CS III. This sequence should be accompanied by the opportunity for additional computing courses based on a variety of factors, including transfer requirements, institutional specializations, and student interests.
>
> Past computer science model curricula have identified at least three “paradigms” or approaches that one could take to computer science content: objects-first (centered on object-oriented programming), breadth-first (an initial holistic view subsequently progressing deeper), and imperative-first (centered on procedural programming). The Computer Science associate-degree transfer program now calls for a blended approach:
>
> - Object-oriented programming is emphasized in CS I, but not necessarily early in the semester.
> - The topics of algorithms and fundamental programming constructs are important components of CS I and are consistent with the Böhm-Jacopini theory for procedural programming.

- The breadth-first approach is used in the coverage of three important topics: ethics and professionalism, security, and software engineering principles.

These topics are covered in deeper and deeper fashion as the student progresses from CS I to CS II to CS III.

Conference on Innovation and Technology in Computer Science Education (ITiCSE) Information Assurance Curriculum Guidelines Working Group Guidelines

The following information is from http://delivery.acm.org/10.1145/1980000/1971686/p49-cooper.pdf?key1=1971686&key2=7516375031&coll=DL&dl=ACM&ip=128.237.28.14&CFID=23747146&CFTOKEN=92148513 [Cooper 2010].

Information assurance and information security are serious worldwide concerns. Computer security is one of the three new focal areas of the ACM/IEEE's Computer Science Curriculum update in 2008. This ACM/IEEE report describes, as the first of its three recent trends, "the emergence of security as a major area of concern.

The purpose of this working group report is to continue the work of the 2009 working group on information assurance (IA) education. The focus of the 2010 working group is to examine the curricula of existing academic programs, as well as at the key academic governmental and industry IA education standards and guidelines identified by the 2009 IA working group in order to begin defining the IA education space as a first step towards developing curricular guidelines.

Survivability and Information Assurance (SIA) Curriculum

The following information comes from http://www.cert.org/sia/ [CERT 2007].

Introduction

Today's organizations rely on networked systems powered by fast-changing technology. This reliance makes them more vulnerable to attacks and forces system administrators to seek new approaches to computer and network security. To help them, CERT has developed a downloadable three-course curriculum in survivability and information assurance (SIA). This curriculum offers a problem-solving methodology built on key SIA principles that are independent of specific technologies. These principles form the foundation of CERT’s SIA Curriculum. A summary of the curriculum is provided below.

SIA Curriculum Foundations

We based the SIA Curriculum on five key foundations. Each is detailed in Foundations of the SIA Curriculum:

1. Principles of Survivability and Information Assurance: Making decisions through an organized thought process
2. The Enterprise Network Supports the Mission of the Business: Understanding how technology choices and applications impact the mission of the business

3. Survivable Functional Units: Reducing the complexity of the enterprise to a manageable size
4. Inherit an Enterprise Network: Integrating seamlessly new functionality in the network while keeping mission and constraints of the business in focus
5. Challenge Assumptions: Understanding first the assumptions, challenging them, and then making an informed decision

These foundations inform the courseware in the SIA Curriculum. Understanding them is the key to successfully teaching and implementing it.

SIA Curriculum Overview

The SIA Curriculum Overview explains the key features of the SIA curriculum: its audience, structure, the technology used, and the characteristics students and teachers should possess to be able to get the most out of the curriculum.

The curriculum consists of the following major topic areas, each of which corresponds to one course:

1. Principles of Survivability and Information Assurance: This course presents in detail the ten principles of survivability and information assurance, on which the entire SIA curriculum is based.
2. Information Assurance Networking Fundamentals: This course applies the ten principles to the concepts and an implementation of TCP/IP networking.
3. Sustaining, Improving, and Building Survivable Functional Units (SFUs)

Computing Curricula 2001

The following information is from *Computing Curricula 2001: Computer Science, Final Report* [IEEE-CS 2001].

CC 2001- Executive Summary

This document represents the final report of the Computing Curricula 2001 project

(CC2001)—a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) to develop curricular guidelines for undergraduate programs in computing. The report continues a long tradition of recommendations for academic programs in computing related fields dating back to 1965, as described in Chapter 2 of the report.

This volume of the report outlines a set of recommendations for undergraduate programs in computer science. As described in Chapter 1, the CC2001 report will eventually consist of several volumes containing separate recommendations for other computing disciplines, including computer engineering, software engineering, and information systems. Those reports are each under the control of separate committees and will be published as they are completed.

Highlights of this report include the following:

- *The CS body of knowledge.* We have identified a body of knowledge appropriate to undergraduate computer science programs. Drawing on the structure of earlier curriculum reports, we have arranged that body of knowledge hierarchically, subdividing the field into areas, which are then broken down further into units and individual topics. An overview of the body of knowledge appears in Chapter 5.
- *The CS undergraduate core.* From the 132 units in the body of knowledge, we have selected 64 that represent core material, accounting for approximately 280 hours of instruction. As noted in our statement of principles in Chapter 4, we defined the core as the set of units for which there is a broad consensus that the material is essential to an undergraduate degree in computer science. The philosophy behind the definition of the core is described in more detail in Chapter 5.
- *Learning objectives.* For each of the units in the body of knowledge, we have developed a set of learning objectives designed to promote assessment of student achievement. These learning objectives appear as part of the detailed description of the body of knowledge in Appendix A. In addition to the individual learning objectives, Chapter 11 of the report outlines a more general set of objectives that all computer science graduates should be able to meet.
- *Curriculum models.* The report identifies six approaches to introductory computer science that have proven successful in practice, as described in Chapter 7. Building on that foundation, Chapter 8 offers a set of four thematic approaches for presenting the core material in intermediate-level courses. The discussion of curricular models continues in Chapter 9, which offers several models for the curriculum as a whole.
- *Course descriptions.* Appendix B contains detailed course descriptions for 47 courses that are part of the various curriculum models. In addition, we have identified over 80 additional advanced courses that would be appropriate for undergraduate programs.

The process of developing the report has been highly inclusive. More than 150 people have been directly involved in the focus groups established to contribute to the process. In addition, the report has been widely reviewed by academics and practitioners through a series of three public drafts. We have also held a series of feedback sessions at conferences and meetings, including the Special Interest Group on Computer Science Education symposium (SIGCSE), the Frontiers in Education conference (FIE), the World Congress on Computers and Education (WCCE), along with various smaller meetings in Europe, Asia, and various parts of the United States. These meetings have provided us with critically important feedback, which we have used to shape the final report.

The length of the introductory sequence

Although the philosophy and structure of introductory courses have varied widely over the years, one aspect of the computer science curriculum has remained surprisingly constant: the length of the introductory sequence. For several decades, the vast majority of institutions have used a two-course sequence to introduce students to computer science. In the computer science education community, these two courses are generally known as CS1 and CS2, following the lead of Curriculum ’78 [ACM78]. While the content of these courses has evolved over time in response to changes in technology and pedagogical approach, the length of the sequence has remained the same.

We believe the time is right to question this two-course assumption. The number and complexity of topics that entering students must understand have increased substantially, just as the problems we ask them to solve and the tools they must use have become more sophisticated. An increasing number of institutions are finding that a two-course sequence is no longer sufficient to cover the fundamental concepts of programming, particularly when those same courses seek to offer a broader vision of the field. Expanding the introductory sequence to three courses makes it far easier to cover the growing body of knowledge in a way that gives students adequate time to assimilate the material.

The CC2001 Task Force strongly endorses the concept of moving to a three-course introductory sequence and believes that this option will prove optimal for a relatively wide range of institutions. At the same time, the three-course approach will not be right for everyone. The fact that the traditional two-course approach fits into a single year of study at semester-based institutions often makes it easier to fit the introductory material into the whole of the curriculum without interfering with the scheduling of sophomore level courses. Similarly, the task of assigning credit for courses taken at other institutions, including advanced placement programs in secondary schools, becomes more complicated if one institution follows a two-semester calendar while the other covers the introductory material in three.

To support both two- and three-course introductions, the CC2001 Task Force has developed both options. . . [IEEE-CS 2001, p. 29]

Computer Science Curriculum 2008

The following information is from *Computer Science Curriculum 2008: An Interim Revision of CS 2001* [ACM 2008].

Preface to the Interim Revision

In recent times, the ACM and the IEEE Computer Society have sought to provide curriculum guidance on computing at approximately ten-year intervals. Thus 1968, 1978, 1991, and 2001 were the dates of publication of previous guidance on Computer Science.

Around the time of the publication of the most recent Computer Science volume, in December 2001, a commitment was made by the ACM and the Computer Society to provide curriculum guidance on a more regular basis. This was to recognize the rapid rate of change in the discipline and the consequent need for guidance to the community. It was felt that after a period of around 5 years steps should be taken to address this. Yet such guidance should not be seen to create revolution or confusion in the community; rather it should help and support. This present volume is provided in that spirit.

Since 2001, much has happened in computing. Today there is talk of a crisis, with enrollments having plummeted in many countries, often by as much as 60 – 70% from the peak of 2001. This fall in numbers has come at a time when there is increased recognition of the role of computing in innovation across engineering, in science, in business, in education, in entertainment and indeed in all walks of life. At the same time, the number of jobs in computing has risen while the supply of good graduates has fallen and some data suggests is failing to meet the demand in certain countries. The reasons for this are many and complex. However, many argue that the traditional curriculum in computing is unattractive to present-day students and that creates a challenge.

Part of the CC 2001 endeavor was to create documents that would complement the Computer Science guidance document. This resulted in the publication, over recent years, of volumes in Computer Engineering, Information Systems, Information Technology, and Software Engineering. An Overview volume has also been published; this sought to highlight the differences and draw out the similarities, but basically to provide a framework within which the various volumes could be seen to fit. This creates a different kind of environment in which to review the Computer Science volume.

Taking all these various matters into consideration, this review of the computer science volume comes at a crucial time. In addition, there is wide recognition that a considerable amount of work is needed to discover better and more effective ways of presenting the discipline of computing. This has enormous importance, economic and strategic.

Yet it would be misleading to recommend ideas that were not regarded as sound advice and best practice based on appropriate trials and testing.

This interim review has benefited from input from many (from industry, academia, etc.) through consultation and through discussion. It should be seen as a necessary updating of the influential CS2001 volume. More precisely, the CS2001 Body of Knowledge has been updated and there is additional commentary / advice in the accompanying text. The process has lead to wide recognition of the need to find new and better ways to present and portray the discipline of computer science; that remains a challenge for us all.

December 2008

Software Assurance Undergraduate Course Outlines

The following is the abstract from *Software Assurance Curriculum Project Volume II: Undergraduate Course Outlines* [Mead 2010b].

Modern society depends on software systems of ever-increasing scope and complexity. Virtually every sphere of human activity is impacted by these systems, from social interaction in our personal lives to business, 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 second volume in the Software Assurance Curriculum Project sponsored by the Department of Homeland Security. The first volume, the Master of Software Assurance Reference Curriculum (CMU/SEI-2010-TR-005), presented a body of knowledge from which to create a Master of

Software Assurance degree program, as both a standalone offering and as a track within existing software engineering and computer science master's degree programs. This report focuses on an undergraduate curriculum specialization for software assurance. The seven courses in this specialization are intended to provide students with fundamental skills for either entering the field directly or continuing with graduate-level education.

Appendix B: Bloom’s Taxonomy and the GSwE2009

Bloom’s Taxonomy is a classification system devised in 1956 by a group of educators lead 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 13 provides a description of the Bloom’s Levels for the Cognitive Domain.

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

Table 13: 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, and 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 C: Community College Profiles

Students at Community Colleges (According to the American Association for Community Colleges)

The information below was taken from http://www.aacc.nche.edu/AboutCC/Trends/Pages/studentsatcommunitycolleges.aspx [AACC 2011].

> Community colleges are a vital part of the postsecondary education delivery system. They serve almost half of the undergraduate students in the United States, providing open access to postsecondary education, preparing students for transfer to 4-year institutions, providing workforce development and skills training, and offering noncredit programs ranging from English as a second language to skills retraining to community enrichment programs or cultural activities.
>
> Community colleges serve close to half of the undergraduate students in the United States, which included more than 6.5 million credit students in the fall of 2010. The comprehensive mission of community colleges makes them attractive to a broad range of people who seek particular programs or opportunities of special interest. Community colleges are the gateway to postsecondary education for many minority, low income, and first-generation postsecondary education students. Since 1985, more than half of all community college students have been women. In addition, the majority of Black and Hispanic undergraduate students in this country study at these colleges.
>
> Community colleges also provide access to education for many nontraditional students who are adults and working while enrolled. The average age of a community college student is 29 years old, and two thirds of community college students attend part-time. At the same time, community colleges are not only providing access for adult students but also serving an increasing number of traditional age and high school students who take specific courses to get ahead in their studies. In fact, half of the students who receive a baccalaureate degree attend community college in the course of their undergraduate studies.
>
> The costs to attain a postsecondary degree are on the rise. As a result, increasing numbers of students at community colleges (and 4-year institutions) are looking to the federal financial aid programs to help offset or finance the costs of their education. Almost half of the students attending community college receive some form of financial aid to help finance their studies. In 2005, more than 2 million community college students received Pell grant dollars. However, in recent years, there has been a shift in government policies away from grants toward student loans. Because of the low costs to attend community college, the amounts borrowed are lower for community college students than they are for their counterparts at 4-year institutions (public and private).
>
> Community colleges are diverse institutions that serve a wide variety of needs. These include the students who come to upgrade their skills for a particular job, students who are pursuing an

associate degree to transfer to a 4-year institution and students who come to pursue a hobby (such as learning a language). The educational outcomes of community college students reflect this diversity.

According to the Community College Research Center located at Columbia University, Teachers College, most community college students are older than 25 though many students are also high school graduates who want to cost-effectively start their college education.

Community College Research Center located at Columbia University, Teachers College

The following information was taken from http://ccrc.tc.columbia.edu/History.asp [CCRC 2011].

The Community College Research Center (CCRC) is the leading independent authority on the nation's more than 1,200 two-year colleges. Since our inception, CCRC's consortium of researchers has strategically assessed the problems and performances of community colleges. Our mission is to conduct research on the major issues affecting community colleges in the United States and to contribute to the development of practice and policy that expands access to higher education and promotes success for all students. CCRC's extensive body of research provides a strong foundation on which to build new policies and initiatives to improve the outcomes of these institutions so integral to the higher education system, employment landscape, and national economy.

Community colleges serve high numbers of non-traditional students who are often older than 25, working part- or full-time, parents with dependent children, from ethnic minorities, and from low-income households. These institutions cater to high school students seeking enhanced learning opportunities, high school graduates looking for a cost-effective way to begin their college education, individuals interested in obtaining technical or vocational certification, and full-time employees who seek special training.

CTE Dual Enrollment: Preparing Students for College and Careers

The following information was taken from http://ccrc.tc.columbia.edu/Presentation.asp?UID=357 [Hughes 2011].

Conference: California Community College Association for Occupational Education (CCCAOE) 2011 Conference

Date: March 23, 2011 4:00PM

Location: Oakland Marriott City Center, Oakland, CA

Additional relevant research articles are summarized in the appendix.

A few scenarios of typical community college "computing" students include: ("computing" in this context is used broadly referring to computer science, information technology, etc.)

- High school students who want to become computer programmers and/or game developers.

  - Typically want to earn associates degree intending either to enter the workforce (A.A.S) or transfer to a baccalaureate program (A.S.).
- Non-computing baccalaureate degree holders who need to learn about computers so they can either apply these skills to their field or enter into a career in "computing."
  - Typically want to earn either associates degree or technical certificate.
- "Computing" baccalaureate degree holders who need to update their knowledge and skills (lifelong learning). For example, the waning programming languages of COBOL and RPG.
  - Typically want to earn a technical certificate or not interested in any credential just the knowledge and skill.
- "Computing" professional in need of a credential to further their career, such as CISSP or Security +.

For our purposes, we narrowed down our audience to include:

- High school students intending to earn an A.S. degree in order to transfer to a 4-year college
- Baccalaureate degree holders with degrees in computing, math, science, or engineering who want to update their skills
- Existing computing professionals in need of a credential to further their career

We do not include high school students who want a two year degree in order to immediately enter the workforce, or baccalaureate degree holders in "non-computing" or non-technical fields in our target audience.

Appendix D: Relevant Research Articles

The authors reviewed the following research articles that are relevant to community college education.

"The Reverse Transfer Process" in *Community College Review* [Chen 2008]

> "These *reverse transfer students* have graduated high school, and they have attended college for a period of time or, in some cases, have even graduated from a traditional four-year college. For a variety of reasons, though, these students decide that the traditional four-year college is just not for them, and they embrace the opportunity to enroll in and to attend a two-year community college.
>
> Subsequently, they transfer from their four-year college and join a two-year college, and while they are moving forward in terms of their education, they are 'taking a step back' by switching from a traditional college or university to a community college. Hence, they are *reverse transfer students*."

"The New Reverse Transfer" in *Inside Higher Ed* [Moltz 2009]

> "Stephanie Jamiot is a community college transfer student, but not the kind one might expect. Instead of following the steady flow of students who move from two-year institutions to four-year institutions, she is one of a growing number of so-called 'reverse transfers' who leave four-year universities to attend community college.
>
> Cuyahoga Community College in Cleveland – Ohio's largest two-year institution and the one Jamiot currently attends – had an 11 percent increase in the number of 'reverse transfers' this spring compared to last. These students mostly come from public and private institutions around Ohio. Nationally, the American Association of Community Colleges notes that a third of all two-year students previously attended a four-year institution. The recession had led to a surge in community college enrollments this year, and some experts believe these 'reverse transfers' are an important and sometimes overlooked portion of that growing student body at two-year institutions."

"**Four-Year Graduates Attending Community Colleges As Serious Credit Students**" by Community College Research Center [Quinley 1988]

> "This [CCRC] Brief is drawn from a report of a qualitative study conducted at Central Piedmont Community College in Charlotte, North Carolina. The study took two approaches – a telephone survey of four-year graduates who had completed at least 15 credit hours at that community college, and an examination of student records to describe the enrollment trends of this population over a ten-year period. The results from these two approaches were compared with the findings reported in the literature.

In the fall of 1996, Central Piedmont Community College had over one thousand students (1,104) – a little more than 7 percent of the school's total enrollment – who had previously earned a baccalaureate degree."

"The College of 2020: Students" by Chronicle Research Services [Chronicle Research Services 2009]

"This is the first Chronicle Research Services report in a three-part series on what higher education will look like in the year 2020. It is based on reviews of research and data on trends in higher education, interviews with experts who are shaping the future of colleges, and the results of a poll of members of a Chronicle Research Services panel of admissions officials.

The traditional model of college is changing, as demonstrated by the proliferation of colleges, hybrid class schedules with night and weekend meetings, and, most significantly, online learning. The idyll of four years away from home—spent living and learning and growing into adulthood—will continue to wane. It will still have a place in higher education, but it will be a smaller piece of the overall picture.

Students' convenience is the future. More students will attend classes online, study part time, take courses from multiple universities, and jump in and out of colleges. Students will demand more options for taking courses to make it easier for them to do what they want when they want to do it. And they will make those demands for economic reasons, too. The full-time residential model of higher education is getting too expensive for a larger share of the American population. More and more students are looking for lower cost alternatives to attending college. Three-year degree programs, which some colleges are now launching, will almost assuredly proliferate. The trend toward low-cost options also will open doors for more inexpensive online options.

Community colleges and for-profit institutions should continue to thrive because of their reputations for convenience. The rest of colleges—regional public universities, small liberal-arts colleges, and private universities without national followings—can expect to compete for students based on price, convenience, and the perceived strengths of the institutions."

Bibliography

URLs are valid as of the publication date of this document.

[AACC 2011]
American Association of Community Colleges (AACC). *Students at Community Colleges*. http://www.aacc.nche.edu/AboutCC/Trends/Pages/studentsatcommunitycolleges.aspx (2011).

[ABET 2010]
ABET, Inc. *ABET: Leadership and Quality Assurance in Applied Science, Computing, Engineering, and Technology Education*. http://www.abet.org/ (2010).

[ACM 1991]
Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS) Joint Curriculum Task Force. *Computing Curricula 1991*. ACM Press and IEEE Computer Society Press (1991).

[ACM 1993]
Association for Computing Machinery (ACM) Two-Year College Computing Curricula Task Force. *Computing Curricula Guidelines for Associate-Degree Programs: Computing Sciences.* ACM Press (1993).

[ACM 1999]
Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS) Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP). *Software Engineering Code of Ethics and Professional Practice (Version 5.2).* ACM & IEEE-CS, 1999. http://www.acm.org/serving/se/code.htm

[ACM 2003]
Association for Computing Machinery (ACM) Two-Year College Computing Curricula Task Force. *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science*. ACM Press (2003).

[ACM 2008]
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]
Association for Computing Machinery (ACM) Committee for Computing Education in Community Colleges. *Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science*. ACM and IEEE Computer Society, 2009. http://www.acmccecc.org/committee/CommitteeFileUploads/2009ComputerScienceTransferGuidelines.pdf

[ACM 2011]
Association for Computing Machinery (ACM) Inc. *ACM Code of Ethics and Professional Conduct*. http://www.acm.org/constitution/code.html (2011).

[ACM CCECC 2009a]
ACM Committee for Computing Education in Community Colleges (CCECC). Program Details: Computer Science. http://www.acmccecc.org/pgm_inventory/programdetail.aspx?pID=38 (2009).

[ACM CCECC 2009b]
ACM Committee for Computing Education in Community Colleges (CCECC). *Assessment Rubric for Course Learning Outcomes: Computer Science I*. http://www.acmccecc.org/reports/report_assessmentRubric.aspx?cID=43 (2009).

[ACM CCECC 2009c]
ACM Committee for Computing Education in Community Colleges (CCECC). *Assessment Rubric for Course Learning Outcomes: Computer Science II*. http://www.acmccecc.org/reports/report_assessmentRubric.aspx?cID=47 (2009).

[ACM CCECC 2009d]
ACM Committee for Computing Education in Community Colleges (CCECC). *Assessment Rubric for Course Learning Outcomes: Computer Science III*. http://www. acmccecc.org/reports/report_assessmentRubric.aspx?cID=98 (2009).

[ACM CCECC 2009e]
ACM Committee for Computing Education in Community Colleges (CCECC). *Program Details: Software Engineering*. http://acmccecc.org/pgm_inventory/programdetail.aspx?pID=40 (2009).

[ACM CCECC 2009f]
ACM Committee for Computing Education in Community Colleges (CCECC). *Course Details: Introduction to Software Engineering*. http://acmccecc.org/course_inventory/coursedetail.aspx?cID=113 (2009).

[Allen 2008]
Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. Chapters 1, 7, and 8 in *Software Security Engineering: A Guide for Project Managers.* Addison-Wesley Professional, 2008.

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

[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).

[Chen 2008]
Chen, Grace. "The Reverse Transfer Process." *Community College Review*. 2008. http://www.communitycollegereview.com/articles/22

[Chronicle Research Services 2009]
Chronicle Research Services. *The College of 2020: Students*. 2009. http://www.compassknowledge.com/wp-content/uploads/2010/04/06-2009-The-2020-Students-Part-1of-3-The-Chronicle-of-HE.pdf

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

[CCRC 2011]
Community College Research Center (CCRC). *History/Mission*. http://ccrc.tc.columbia.edu/History.asp (2011).

[Cooper 2010]
Cooper, Stephen; Nickell, Christine; Pérez, Lance C.; Oldfield, Brenda; Brynielsson, Joel; Gencer Gökce, Asim; Hawthorne, Elizabeth K.; Klee, Karl J.; Lawrence, Andrea; & Wetzel, Susanne. *Towards Information Assurance (IA) Curricular Guidelines* (ITiCSE 2010 Working Group Report). ACM, 2010. http://delivery.acm.org/10.1145/1980000/1971686/p49-cooper.pdf?key1=1971686&key2=6295195031&coll=DL&dl=ACM&ip=128.237.28.14&CFID=22435773&CFTOKEN=11620354

[CyberWatch 2011]
CyberWatch. *College Curriculum*. http://www.cyberwatchcenter.org/index.php?option=com_content&view=article&id=99&Itemid=64 (Accessed June 2011). Note: Downloading the curriculum requires registration.

[DHS 2011a]
Department of Homeland Security (DHS) Software Assurance (SwA). *Build Security In*. https://buildsecurityin.us-cert.gov/daisy/bsi/home.html (2011).

[DHS 2011b]
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 (2011).

[Hughes 2011]
Hughes, Katherine. “CTE Dual Enrollment: Preparing Students for College and Careers.” Presented at the *CA Community College Association for Occupational Education Conference*. 2011.
http://ccrc.tc.columbia.edu/DefaultFiles/SendFileToPublic.asp?ft=pdf&FilePath=c:\Websites\ccrc_tc_columbia_edu_documents\332_914.pdf&fid=332_914&aid=47&RID=914&pf=Publication.asp?UID=914

[Huitt 2006]
Huitt, W. “The cognitive system.” *Educational Psychology Interactive*. Valdosta State University, http://www.edpsycinteractive.org/topics/cogsys/cogsys.html (2006).

[IEEE-CS 2001]
IEEE Computer Society (IEEE-CS) & Association for Computing Machinery (ACM). *Computing Curricula 2001: Computer Science, Final Report*.
http://www.acm.org/education/curric_vols/cc2001.pdf (2001).

[IEEE-CS 2004a]
IEEE Computer Society (IEEE-CS) & 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) & 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 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).

[IEEE-CS 2011]
IEEE Computer Society. *IEEE Computer Society Educational Activities Board*.
http://www.computer.org/education/ (2011).

[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.

[Mead 2010a]
Mead, Nancy R.; Allen, Julia H.; Ardis, Mark; Hilburn, Thomas B.; Kornecki, Andrew J.; Linger, Rick; & McDonald, James. *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum* (CMU/SEI-2010-TR-005). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr005.cfm

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

[Mead 2011]
Mead, Nancy R.; Allen, Julia H.; Ardis, Mark; Hilburn, Thomas B.; Kornecki, Andrew J.; & Linger, Rick. *Software Assurance Curriculum Project Volume III: Master of Software Assurance Course Syllabi* (CMU/SEI-2011-TR-013). Software Engineering Institute, Carnegie Mellon University, 2011. http://www.sei.cmu.edu/library/abstracts/reports/11tr013.cfm

[Merkow 2010]
Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development*. CRC Press, 2010 (ISBN: 9781439826966).

[Moltz 2009]
Moltz, David. "The New Reverse Transfer." *Inside Higher Ed*. 2009.
http://www.insidehighered.com/news/2009/02/18/reverse

[Quinley 1988]
Quinley, John W. & Quinley, Melissa P. *Four-Year Graduates Attending Community Colleges: A New Meaning for the Term "Second Chance."* Community College Research Center, 1988.
http://ccrc.tc.columbia.edu/DefaultFiles/SendFileToPublic.asp?ft=pdf&FilePath=c:\Websites\ccrc_tc_columbia_edu_documents\332_41.pdf&fid=332_41&aid=47&RID=41&pf=Publication.asp?UID=41

[Seacord 2005]
Seacord, Robert C. *Secure Coding in C and C++*. Addison-Wesley, 2005.

[Stallings 2007]
Stallings, W. *Network Security Essentials*, 3rd Ed. Prentice Hall, Upper Saddle River, NJ, 2007.

[Stevens 2004]
Stevens, Danielle D. & Levi, Antonia J. *Introduction to Rubrics: An Assessment Tool to Save Grading Time, Convey Effective Feedback, and Promote Student Learning*, Stylus Publishing, Virginia (2004).

[Redwine 2010]
Redwine, Samuel T., Jr. *Secure Software Engineering Education*. https://buildsecurityin.us-cert.gov/swa/downloads/JMU_SSE.pdf (Accessed August 2010).

[Stoneburner 2004]
Stoneburner Gary; Hayden,Clark; & Feringa, Alexis. "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A (NIST Special Publication 800-27 Rev A)." *Computer Security*. Computer Science Division, Information Technology Laboratory, National Institute of Standards and Technology, 2004.
http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf

[Taylor 2008]
Taylor, B. & Shiva Azadegan, "Moving beyond security tracks: Integrating security in CS0 and CS1." Presented at *Special Interest Group Computer Science Education (SIGCSE)*. ACM, 2008.

[Wright 2006]
Wright, Marie & Kakalik, John. *Information Security: Contemporary Cases*. Jones & Bartlett Publishers, 2006.

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** September 2011		3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Software Assurance Curriculum Project Volume IV: Community College Education			5. **FUNDING NUMBERS** FA8721-05-C-0003
6. **AUTHOR(S)** Nancy R. Mead, Elizabeth K. Hawthorne, and Mark Ardis			
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-2011-TR-017
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116			10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-2011-017
11. **SUPPLEMENTARY NOTES**			
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS			12B **DISTRIBUTION CODE**
13. **ABSTRACT (MAXIMUM 200 WORDS)** The fourth volume in the Software Assurance Curriculum Project led by a team at the Software Engineering Institute, this report focuses on community college courses for software assurance. The report includes a review of related curricula, outcomes and body of knowledge, expected background of target audiences, and outlines of six courses. The courses are intended to provide students with fundamental skills for continuing with graduate-level education or to provide supplementary education for students with prior undergraduate technical degrees who wish to become more specialized in software assurance. Previous volumes of this project are *Volume I: Master of Software Assurance Reference Curriculum*, *Volume II: Undergraduate Course Outlines*, and *Volume III: Software Assurance Course Syllabi.*			
14. **SUBJECT TERMS** software assurance, software assurance education, software engineering education, software security education, community college education			15. **NUMBER OF PAGES** 65
16. **PRICE CODE**			
17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102