Association for Computing Machinery

Committee for Computing Education in Community Colleges (CCECC)

Curricular Guidance for Associate-Degree Transfer Programs in Computer Science

StrawDog draft version
June 2016

Available for Public Review and Comment from June 30 to July 31, 2016
Feedback being collected at www.surveymonkey.com/r/CS-Cyber-Feedback

Table of Contents

1. Acknowledgements
2. Overview
    a. Introduction
    b. Two-Year/Community College Environment
    c. Cybersecurity across the Computing Curriculum
    d. Ethics and Professionalism across the Computing Curriculum
    e. Internationalization
    f. Assessment
    g. Articulation
    h. Transfer Programs
    i. Career Programs
    j. Teaching and Learning Strategies
    k. Computing Laboratory Experiences
3. The ACM/IEEE CS2013 Body of Knowledge (Abridged version)
    a. Algorithms and Complexity (AL)
    b. Architecture and Organization (AR)
    c. Computational Science (CN)
    d. Discrete Structures (DS)
    e. Graphics and Visualization (GV)
    f. Human-Computer Interaction (HCI)
    g. Information Assurance and Security (IAS)
    h. Information Management (IM)
    i. Networking and Communications (NC)
    j. Operating Systems (OS)
    k. Programming Languages (PL)
    l. Software Development Fundamentals (SDF)
    m. Software Engineering (SE)
    n. System Fundamentals (SF)
    o. Social Issues and Professional Practice (SP)
4. Assessment Metrics
5. Bloom's Revised Taxonomy
6. Glossary of Terms
7. Bibliography

Acknowledgements

The members of the ACM Committee for Computing Education in Community Colleges (CCECC) acknowledges and thanks the ACM Education Board for providing funding for this important curricular project to develop associate-degree curricular guidance in computer science with contemporary cybersecurity concepts (CS-Cyber).

ACM CCECC Members

- Dr. Elizabeth K. Hawthorne, Union County College, NJ
- Dr. Cara Tang, Portland Community College, OR
- Prof. Cindy S. Tucker, Bluegrass Community and Technical College, KY

Computer Science-Cybersecurity Team Leaders

- Prof. Teresa T. Moore, Volunteer State Community College, TN
- Prof. Lambros Piskopos, Wilbur Wright College, IL
- Dr. Christian Servin, El Paso Community College, TX

Computer Science-Cybersecurity Task Force Members

- Prof. Kimberly Bertschy, Northwest Arkansas Community College, AR
- Prof. Colleen Case, Schoolcraft College, MI
- Dr. Markus Geissler, Cosumnes River College, CA
- Dr. Becky Grasser, Lakeland Community College, OH
- Prof. Charles Hardnett, Gwinnett Technical College, GA
- Prof. Amardeep Kahlon, Austin Community College District, TX
- Prof. James Kolasa, Bluegrass Community and Technical College, KY
- Dr. Shamsi Moussavi, MassBay Community College, MA
- Prof. Pam Schmelz, Ivy Tech Community College, IN
- Prof. Melissa Stange, Lord Fairfax Community College, VA
- Prof. Khallai Taylor, Miami-Dade College, FL
- Prof. Carole Tharnish, Central Community College, NB

Other Contributors

- Dr. Anne Applin, Southern Maine Community College, ME
- Prof. Bryce Barrie, Saskatchewan Polytechnic, Canada
- Prof. Michael Bauer, Leeward Community College, HI
- Prof. Paul Dadosky, Ivy Tech Community College, IN
- Prof. Andrea DeMott, Ohio University, OH
- Dean Jamie Edwards, Wytheville Community College, VA
- Prof. Rafael Escalante, El Paso Community College, TX
- Dr. Larry Forman, San Diego City College, CA
- Prof. Dianne Hill, Jackson College, MI

- Dean Nancy Jones, Coastline Community College, CA
- Prof. Marc Nester, Wytheville Community College, VA
- Dr. Dean Nevins, Santa Barbara City College, CA
- Dr. Michael Posner, Villanova University, PA
- Prof. Kristopher Roberts, Ivy Tech Community College, IN
- Prof. Barry Sullens, Ivy Tech Community College, IN
- Prof. Robert Surton, Columbia Gorge Community College, OR

Overview

Introduction

This initial draft, called ***StrawDog***, is an update to the *Guidelines for Associate-Degree Transfer Curriculum in Computer Science,* published by the ACM Committee for Computing Education in Community College (CCECC) in 2009 (ccecc.acm.org/guidance/computer-science). Over 30 community college and university computing educators contributed to the creation of **StrawDog** as either members of the CS-Cyber task force who met virtually in teams over the course of months or through participating in a half day workshop at SIGCSE 2016 in Memphis, TN. We continue to engage the community by asking for your constructive feedback on ***StrawDog.*** Your feedback is SO important in making the next draft version and ultimately the final version better and more useful to members of our computing education community. The members of the ACM CCECC thank you for taking the time to provide your comments via www.surveymonkey.com/r/CS-Cyber-Feedback.

The professional societies of the ACM and the IEEE Computer Society have a long history of collaborating on computing materials for higher education. These organizations have jointly produced significant volumes of curricular recommendations and guidelines for associate, baccalaureate and graduate computing programs; these volumes are referred to as the ACM Computing Curricula series (www.acm.org/education/curricula-recommendations). Likewise, the ACM Committee for Computing Education in Community Colleges (CCECC) has produced a corresponding set of curricular guidelines that provide similar guidance for associate-degree granting institutions, in a manner that fosters inter-institutional cooperation and student articulation. This model curriculum provides discussion on transfer considerations and discussion on articulation in computer science.

Like most countries, cybersecurity is a national priority in the United States with a formal action plan (www.whitehouse.gov/the-press-office/2016/02/09/fact-sheet-cybersecurity-national-action-plan). Higher education is a key component of the cybersecurity national action plan (NCAP). ACM responded by integrating information assurance and security learning outcomes throughout its most current computer science curricular guidance, CS2013. In 2015, ACM in association with the Computer Society of the Institute of Electrical and Electronics Engineerings (IEEE-CS), the Association for Information Systems (AIS), the Cyber Education Project (CEP), and the International Federation of Information Processing (IFIP) formed a joint task force to create undergraduate curricular guidelines in cybersecurity (www.csec2017.org). In similar fashion and informed by the *National Cybersecurity Workforce Framework* (csrc.nist.gov/nice/framework/), the ACM CCECC is integrating contemporary cybersecurity content into the revision of its 2009 *Guidelines for Associate-Degree Transfer Curriculum in Computer Science*.

Two-Year/Community College Environment

According to the American Association of Community Colleges, nearly one-half of all undergraduates in the United States are enrolled in two-year colleges, and more than half of all first-time college freshman attend community and technical colleges. “Community colleges are

centers of educational opportunity. They are an American invention that put publicly funded higher education at close-to-home facilities, beginning nearly 100 years ago with Joliet Junior College (in Joliet, Illinois). Since then, they have been inclusive institutions that welcome all who desire to learn, regardless of wealth, heritage, or previous academic experience. The process of making higher education available to the maximum number of people continues to evolve" (http://www.aacc.nche.edu/).

The community college environment is uniquely positioned, resulting from the threefold mission of these institutions to provide a learning environment for:

1) transfer into baccalaureate programs;
2) entrance into the local workforce; and
3) lifelong learning for personal and professional enrichment.

In addition, many two-year colleges are drivers of local economic development, providing workforce development and skills training, as well as offering noncredit programs ranging from English as a second language to skills retraining to community enrichment programs or cultural activities.

Two-year colleges serve high school graduates proceeding directly into college, workers needing to upgrade skill sets or master new ones in order to re-enter the workforce, immigrants seeking to become integrated into the local culture and master a new language, individuals leaving the workplace to engage college-level coursework for the first time, returning students with college degrees who have decided to pursue an alternate career path, and many individuals in need of ongoing training and skill updating. This diversity is addressed in numerous ways, including targeted career counseling, remediation of basic skills, specialized course offerings, individualized instruction and attention, flexible scheduling and delivery methodologies, and a strong emphasis on retention and successful completion. Furthermore, because two-year colleges have less restrictive entrance requirements, faculty must be prepared to instruct students exhibiting a broad range of academic preparations, aptitudes, and learning styles. The mission of two-year college faculty is to focus their full-time attention on effective pedagogy for educating a diverse student population, remaining current in their discipline and in the scholarship of teaching and learning, and fostering student success.

Two-year, community or technical colleges, as well as certain four-year colleges, award associate degrees to students completing between 60 to 66 higher education credits in a particular program of study. It is often the case that an associate-degree requires approximately half the college credit of a bachelor's degree. Associate-degree programs are complete in their own right, whether designed specifically to enable graduates to transfer into the upper division of a baccalaureate program or to gain entry into the workforce. These institutions also offer certificate programs, intended to be fulfilled in less time than a complete degree program; such programs are often designed for targeted student audiences and focused on specific content.

At the earliest opportunity, faculty and academic advisors must help each student determine which type of program best serves the student's educational and career goals. Such considerations include the distinctions between certificate, career and transfer programs, the academic requirements of each, and the associated employment options. Career-oriented associate-degree (typically AAS) programs provide the specific knowledge, skills, and abilities (KSA) necessary to proceed directly into the workplace, while transfer-oriented degree (typically AS) programs provide the academic foundation and pathway to continue a program of study at a four-year college or university.

Cybersecurity across the Computing Curriculum

Whether referred to as "cybersecurity", "computer security", "information security", "information assurance" or some other name, curriculum content in creating and maintaining secure computing environments is a critical component in associate-degree computing programs.

Almost every career path open to a computing student encompasses some aspect of security. System administrators and engineers must be able to properly design, configure and maintain a secure system; programmers and application developers must know how to design and build secure, fault-tolerant software systems from the bottom up; web specialists must be capable of assessing risks and determining how best to reduce the potential impact of breached systems; user support technicians must be knowledgeable in security concerns surrounding desktop computing; and project managers must be able to calculate the cost/benefit tradeoffs involved with implementing secure systems.

It is the responsibility of faculty to ensure that students are well prepared for the security challenges they will inevitably encounter in their careers as computing professionals. This can be addressed by way of a variety of implementation strategies. One approach that some associate-degree computing programs offer is a host of individual courses on specific security topics. This approach can provide a wealth of content opportunities for specialization but may create scheduling challenges for many students as it runs the risk of students graduating without having taken sufficient electives to achieve the understanding of the security concepts necessary to function in their professional roles.

Another approach is to fully integrate and incorporate contemporary cybersecurity content into core, introductory computer science courses with specialized courses reserved for targeted settings. These guidelines employ the integrated approach, incorporating relevant cybersecurity student learning outcomes throughout the computer science body of knowledge for lower division, undergraduate curriculum. The Committee also strongly advocates for learning activities that require students to actively demonstrate mastery of the tenets of professional conduct, ethical and responsible behavior, as well as an appreciation for cybersecurity in a holistic manner.

Ethics and Professionalism across the Computing Curriculum

Professional, legal and ethical issues are important elements in the overall curricula for computing disciplines, and must be integrated throughout the programs of study. This context should be established at the onset and these matters should appear routinely in discussions and learning

activities throughout the curriculum. The ACM Code of Ethics notes that “When designing or implementing systems, computing professionals must attempt to ensure that the products of their efforts will be used in socially responsible ways, will meet social needs, and will avoid harmful effects to health and welfare.” (www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct). The Code goes on to provide an excellent framework for conduct that should be fostered beginning early in students’ experiences.

As computing technologies become ubiquitous in society, ethical behavior and adherence to codes of conduct for computing professionals is imperative; therefore, careful consideration of legal, ethical, and societal issues involving computing, the Internet and databases are essential to the education of computing professionals. Students who realize the potential uses and abuses of technology will, as citizens, be able to contribute to public policy debate from a knowledgeable perspective on issues such as property rights and privacy concerns that affect everyone.

Computer systems have substantial social impact in nearly every setting including applications such as healthcare, finance, transportation, defense, government, education, and communications; real-time and safety-critical systems typically have acceptable margins of error close to nil. Developers and support technologists of such computing systems are confronted by challenges regarding choices and tradeoffs in the design, implementation, and maintenance of these systems. Engaging students in the consideration of the ethical aspects for such decisions as well as giving them practice in identifying and weighing the ethical issues enables them to make more judicious choices. It is crucial that students pursuing computing careers be made aware of and properly equipped to handle the complexities of professional judgments; as computing professionals, graduates must follow codes of conduct and take responsibility for their actions and be accountable for the systems that they develop and support.

Internationalization

In the process of developing its curricular guidance in computer science, the ACM CCECC continually seeks international perspectives from two-year post-secondary (“tertiary”) higher education programs both locally within the United States and globally throughout the world. Feedback on this early draft is being sought from countries around the globe, such as China, India, Turkey, South Africa, Australia, New Zealand, Canada, Mexico, Peru, Brazil, the United Kingdom, and countries across the European Union.

Furthermore, the American Association of Community Colleges (AACC) has established an Office of International Programs and Services whose stated goals are “to advocate the community college role in global education among key constituencies, nationally and internationally, to advance global exchanges and partnerships between member colleges and international entities, and to promote intercultural understanding and engagement among students, faculty, staff, and decision makers.” James McKenney, AACC Vice-President for Economic Development and International Programs, spoke as early as 2002 in a reflective interview titled “The Global Linkage” appearing in the journal “US Society and Values” about the rapidly expanding phenomenon of “community colleges”, “two-year technical colleges”, and the “two-year structure” in general throughout Europe, the Americas and Asia.

This phenomenon has since been reported on, promoted and codified in any number of publications and resources. For example, the Council for Industry and Higher Education (CIHE) and The Mixed Economy Group in England have prepared a comprehensive report titled "Higher Education and Colleges: A Comparison Between England and the USA". The United Nations-affiliated Institute of International Education provides a wealth of resources regarding post-secondary and higher education around the world, including activities of institutions akin to the two-year colleges of the United States. The Paris-based Organization for Economic Cooperation and Development (OECD) provides background information, analyses and recommendations for "opportunities for education in the years after compulsory schooling" across Europe, as does the Directorate-General for Education and Culture of the European Commission. The "University World News" and the "World Education News and Reviews" publications are sources of current events worldwide in post-secondary education; the Community College Research Center at Columbia University also provides links to such documentation. The Association of Canadian Community Colleges (ACCC) offers a wealth of information on two-year colleges in Canada. The Ministry of Education of the People's Republic of China describes institutions similar to community colleges in its discussion on "2 to 3-year higher vocational education with the emphasis on high-level professional technical talents."

Assessment

The cognitive outcomes in this guidance are clustered into related Knowledge Areas (KAs). The KAs are subdivided into Knowledge Units (KUs) with a list of corresponding student learning outcomes. Each learning outcome is accompanied by an assessment rubric. The ACM CCECC uses a structured assessment template comprised of three tiers: "emerging", "developed", and "highly developed." Typically as the level of student achievement progresses from "emerging" to "highly developed", the level of Bloom's verbs also increases from the lower order thinking skills (LOTS) to the higher order thinking skills (HOTS). (For sample three-tiered rubrics, see section on *Assessment Metrics*.)

Articulation

Articulation is a key consideration in associate-degree programs which are designed as transfer curricula. Articulation of courses and programs between academic institutions is a process that facilitates transfer by students from one institution to another. The goal is to enable students to transfer in as seamless a manner as possible. Efficient and effective articulation requires accurate assessment of courses and programs as well as meaningful communication and cooperation. Both students and faculty have responsibilities and obligations for successful articulation. Ultimately, students are best served when educational institutions establish well defined articulation agreements that actively promote transfer.

Articulation agreements often guide curriculum content as well, and are important considerations in the formulation of transfer-oriented programs of study. Institutions are encouraged to work collaboratively to design compatible and consistent programs of study that enable students to transfer, in the United States from associate-degree programs into baccalaureate-degree programs, and in other countries from post-secondary colleges into universities. A two-year

college must develop transition and articulation strategies for the colleges and universities to which its students most often transfer, recognizing that it may be necessary to modify course content to facilitate transfer credit and articulation agreements. A program of study must also take into consideration the general education requirements at both the initial college and the anticipated transfer institution. Faculty must ensure that they clearly define program goals, address program learning outcomes, and evaluate students effectively against defined course outcomes. Articulation agreements should specify one or more well-defined exit points for students to matriculate from the post-secondary college to the transfer institution. In turn, faculty at the receiving institution must provide any transitional preparation necessary to enable transfer students to continue their academic work on par with students at their institution. Hence, students must expect to complete programs in their entirety up to well-defined exit points (e.g., completion of a defined course sequence or program) at one institution before transferring to another institution; one cannot expect articulation to accommodate potential transfers in the middle of a carefully designed curriculum. Acting on these considerations, all post-secondary institutions of higher education will foster student success and best serve their students' academic and career aspirations.

Many associate-degree granting institutions have established articulation ("2+2") agreements between their associate degree career programs and corresponding high school programs. For example, many two-year colleges award up to 12 credit hours toward an associate degree in Information Technology to students who complete the Cisco Networking program in high school. In the United States, this is also referred to as middle college.

Transfer Programs

Typically associate-degree computing programs fall into two categories: those designed for transfer into baccalaureate-degree programs (Associate in Science) and those designed to prepare graduates for immediate entry into career paths (Associate in Applied Science). Colleges should make students aware at the onset of their studies of the distinctions between career and transfer programs, the academic requirements of each, and the resultant employment options. Transfer-oriented associate-degree programs rely on formal inter-institutional articulation agreements to ensure that students experience a seamless transition between lower division associate-degree coursework and upper division baccalaureate-degree coursework. Articulation of courses and programs between two academic institutions facilitates the transfer of students from one institution to the other. Faculty and students alike have responsibilities and obligations to achieve successful articulation.

Efficient and effective articulation requires a close evaluation of well-defined course and program outcomes as well as meaningful communication and cooperation. For example, a particular course in one institution might not be equivalent to a single course at a second institution; however, a group or sequence of courses could be determined equivalent to another course grouping or sequence. Faculty must ensure that they clearly define program requirements, address program goals in a responsible manner, and assess students effectively against defined standards. When specifying points of exit within the articulation agreement document, faculty at

the transferring institution must provide sufficient material to prepare students to pursue further academic work at least as well as students at the second institution.

It is not uncommon for students to complete an associate-degree program of study, choose to work for a period of time, and then return to college to pursue their upper division studies for career advancement. (And many employers will provide tuition reimbursement for workers who wish to continue toward a baccalaureate degree.) Because of the ever evolving nature of computing, students must be aware that course content and program requirements are updated frequently, potentially subjecting them to new program requirements and revised articulation agreements. Students are best served when sequences of courses are completed as a unit at one institution due to the comprehensive and conceptual nature of the computing and mathematics content. Hence, students should complete programs of study in their entirety up to well-defined exit points at one institution before transferring to another institution; articulation cannot be expected to accommodate potential transfers in the middle of a well-defined and recognized body of knowledge.

Academic institutions are advised to work collaboratively to design compatible and consistent programs of study that enable students to transfer easily from associate-degree programs into baccalaureate-degree programs. In support of this goal, the ACM provides curricular guidelines for both associate- and baccalaureate-degree programs in computer science.

Career Programs

Typically associate-degree programs fall into two categories: those designed to prepare graduates for immediate entry into career paths and those designed for transfer into baccalaureate-degree programs. Colleges should make students aware at the beginning of their studies of the distinctions between career and transfer programs, the academic requirements of each, and the resultant employment options. Students graduating from a career-oriented associate-degree computing program will typically enter the workforce directly upon graduation.

Career-oriented associate-degree programs provide students with the specific knowledge, skills and abilities (KSA) necessary to proceed directly into employment in a targeted work environment. The program of study will include professional development coursework as well as courses that emphasize communication skills, mathematical reasoning and other general education requirements. The degree granted upon completion of a career-oriented program is typically an Associate in Applied Science (AAS). In addition, many students will augment their formal studies with technical certifications to enhance their immediate employability.

The following factors support the viability of a career-oriented associate-degree program and help ensure the success of students in the workplace:

- An active industry advisory committee consisting of prospective employers, providing guidance concerning the knowledge, skills, and abilities students must possess to enter directly into a career within their community.

- Real-world work experience including co-op programs, internships and other practicum activities, with an emphasis on professional practices. Core and elective coursework as recommended by advisory committees.
- Integration of technical, communication and time-management skills, team projects, and other interpersonal skills that prepare the student for a business working environment.
- Potential articulation paths that enable the career-oriented student to pursue a baccalaureate degree in the future after working for some period of time.
- Assessment processes whereby students can earn credit for relevant experience.

It is important to note that a career-oriented associate-degree program is not intended to facilitate transfer into a baccalaureate program, but rather to provide entry into a career that requires specialized post-secondary skills and an advanced level of expertise and education. Nevertheless, many students graduating from career-oriented programs subsequently select to further their education at the baccalaureate level (frequently with employer tuition assistance plans).

Teaching and Learning Strategies

It is important to engage students' innate interests early in their academic careers to cement their commitment to computing, to further student retention, and to motivate achievement in their coursework. In addition to specific program content, curriculum designers must give consideration to learning activities, instructional techniques and student success. There are specific techniques that can be incorporated that reflect the nature of the work of computing professionals. Activities should be designed so that students learn to work in teams and in the context of projects, gain insights into the real-world setting and associated considerations, see both theory and application, and appreciate the role of foundation material in setting the stage for intermediate topics.

Faculty at two-year colleges must remain aware of the importance of incorporating professional practices and applied work as an integral part of all computing programs. Computing students should be encouraged to:

- work in teams;
- use techniques of task and time management;
- solve practical problems in course projects;
- make presentations;
- confront issues of privacy, confidentiality and ethics;
- use current technology in laboratories;
- attain real-world experience through cooperative education, internships, and/or other practicum activities; and
- participate in student chapters of computing societies and organizations for professional development opportunities.

Increasingly, the area of computing has become critical to the operation of many organizations. Colleges should ensure that students are familiar with the nature of this field and the expectations of the workplace. An active industry advisory committee is an important asset in helping faculty incorporate current professional practices into the curriculum. Computing employees must demonstrate professionalism and ethical behavior, adhere to codes of conduct, safeguard

confidentiality, and respect privacy. They must take responsibility for their actions, be accountable to the organization, understand the impact of their work on others, and demonstrate effective and efficient work practices. This field also demands that professionals engage in an ongoing process of professional growth and development to ensure that their skills and abilities remain current with ever-changing technology. Faculty know that a conscious and proactive incorporation of professional practices into a computing curriculum benefits students, either as a valuable component in a transfer-oriented program, or in addressing industry needs for qualified personnel as they exit a career-oriented program.

Computing Laboratory Experiences

The computer laboratory experience is an essential part of the computing curriculum, either as an integral part of a course or as a separate stand-alone course. Such experiences should start early in the curriculum, the very beginning when students are often motivated by the “hands-on” nature of computing. Introductory laboratories should be designed and conducted to reinforce concepts presented in lecture classes and homework. Students should be provided many opportunities to observe, explore and manipulate characteristics and behaviors of actual devices, systems, and processes. Every effort should be made by instructors to create excitement, interest and sustained enthusiasm in computing students. Many associate-degree granting institutions will be familiar with strong lab-based learning activities, drawing on years of experience with programs such as electronics technology and industry-provided networking curricula. Numerous colleges have long recognized that experiences such as survey courses in engineering often engage students in stimulating activities that peak their interests and set the stage for career choices in such fields. These colleges will find that they can leverage existing facilities, resources and faculty expertise in implementing computing programs.

The ACM/IEEE CS2013 Body of Knowledge
(Abridged Version)

The ACM/IEEE CS2013 Body of Knowledge (BoK) serves as the foundational curricular framework for these associate-degree transfer guidelines in computer science. The CS2013 BoK is organized into a set of 18 Knowledge Areas (KA) that correspond to topical areas of study in computing for undergraduate, baccalaureate degree programs in computer science. These associate-degree transfer guidelines include 15 of the 18 Knowledge Areas, purposefully excluding the Intelligent Systems, Parallel and Distributed Computing, and Platform-based Development KAs, each for different reasons. The Intelligent Systems KA consists mostly of elective content that is more appropriate for upper division undergraduate instruction. The learning outcomes in the Systems Fundamentals and Operating Systems KAs that are part of these associate-degree guidelines cover the appropriate introductory content in the Parallel and Distributed KA for lower division, computer science transfer curriculum. Lastly, there are no core hours associated with the Platform-based Development (PBD) KA, but rather PDB "is concerned with the design and development of software applications that reside on specific software platforms. In contrast to general purpose programming, platform-based development takes into account platform-specific constraints. For instance, web programming, multimedia development, mobile computing, app development, and robotics are examples of relevant platforms that provide specific services/APIs/hardware that constrain development." (cs2013.org, p. 144).

The 15 Knowledge Areas of the Associate-Degree Transfer Curriculum

1. Algorithms and Complexity (AL)
2. Architecture and Organization (AR)
3. Computational Science (CN)
4. Discrete Structures (DS)
5. Graphics and Visualization (GV)
6. Human-Computer Interaction (HCI)
7. Information Assurance and Security (IAS)
8. Information Management (IM)
9. Networking and Communications (NC)
10. Operating Systems (OS)
11. Programming Languages (PL)
12. Software Development Fundamentals (SDF)
13. Software Engineering (SE)
14. System Fundamentals (SF)
15. Social Issues and Professional Practice (SP)

Below are the learning outcomes organized within the 15 Knowledge Areas (KA) and associated Knowledge Units (KU) that together comprise ACM's associate-degree transfer curriculum in computer science. The learning outcomes are expressed using action verbs from

Bloom's Revised Taxonomy (http://ccecc.acm.org/assessment/blooms). The Bloom's level is indicated in *[italics]* after each learning outcome.

Algorithms and Complexity (AL) Knowledge Area

AL/Basic Analysis KU Learning Outcomes

1) Explain the differences among best, average, and worst case behaviors of an algorithm. *[Understanding]*
2) Calculate informally the time and space complexity of simple algorithms. *[Applying]*
3) Explain Big O notation. *[Understanding]*
4) Classify algorithms based upon their Big O notation. *[Understanding]*
5) Contrast standard complexity classes. *[Analyzing]*
6) Execute algorithms on input of various sizes and compare performance. *[Applying]*
7) Analyze time and space trade-offs in algorithms. *[Analyzing]*
8) Explain the importance of writing secure and robust programs. *[Understanding]*

AL/Algorithmic Strategies KU Learning Outcomes

1) List examples of direct and indirect information flows. *[Remembering]*
2) Identify a variety of data structures including stacks, queues, priority queues, references, linked structures, resizable arrays, trees, graphs, and hash tables. *[Remembering]*
3) Demonstrate how to access certain positions of an array. *[Understanding]*
4) Demonstrate which data model (list, set, hash table, tree, or graph) is appropriate for solving a problem. *[Understanding]*
5) Compare and contrast approaches for resolving collisions in hash tables, e.g., linear probing, quadratic probing, double hashing, rehashing, and chaining. *[Analyzing]*
6) Assess secure programming by writing a program that checks boundaries in an array as an example of a buffer over/underflow. *[Evaluating]*
7) Implement a recursive solution to a problem. *[Applying]*

AL/Fundamental Data Structures and Algorithms KU Learning Outcomes

1) Implement basic numerical algorithms. *[Applying]*
2) Implement simple search algorithms and explain the differences in their time complexities. *[Applying]*
3) Implement common quadratic and O(N log N) sorting algorithms. *[Applying]*
4) Describe the implementation of hash tables, including collision avoidance and resolution. *[Understanding]*
5) Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing. *[Understanding]*
6) Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data. *[Understanding]*
7) Explain how tree balance affects the efficiency of various binary search tree operations. *[Understanding]*

8) Solve problems using fundamental graph algorithms, including depth-first and breadth-first search. *[Applying]*

AL/Basic Automata, Computability and Complexity KU Learning Outcomes

1) Use a regular expression to represent a specified language. *[Applying]*
2) Describe a deterministic finite state machine to accept a specified language. *[Understanding]*
3) Explain the role of random numbers in security, beyond just cryptography (e.g. password generation, randomized algorithms to avoid algorithmic denial of service attacks). *[Understanding]*
4) Discuss the concept of finite state machines. *[Understanding]*
5) Explain why the halting problem has no algorithmic solution. *[Understanding]*

Architecture and Organization (AR) Knowledge Area

AR/Machine Level Representation of Data KU Learning Outcomes

1) Explain why everything is data, including instructions, in computers. *[Understanding]*
2) Explain reasons for using alternative formats to represent numerical data. *[Understanding]*
3) Explain how fixed-length number representations affect accuracy and precision. *[Understanding]*
4) Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays. *[Understanding]*
5) Convert numerical data from one format to another, such as, negative integers into sign-magnitude and two's-complement representations. *[Applying]*

AR/Assembly Level Machine Organization KU Learning Outcomes

1) Explain the organization of the classical von Neumann machine and its major functional units. *[Understanding]*
2) Demonstrate how high-level language patterns map to assembly/machine language, including subroutine calls. *[Understanding]*
3) Explain the basic concepts of interrupts and I/O operations. *[Understanding]*
4) Write simple assembly language program segments. *[Applying]*

AR/Memory System Organization and Architecture KU Learning Outcomes

1) Identify the main types of memory technology and their relative cost and performance. *[Remembering]*
2) Explain the effect of memory latency on running time. *[Understanding]*
3) Describe how the use of memory hierarchy is used to reduce the effective memory latency. *[Understanding]*

Computational Science (CN) Knowledge Area

CN/Introduction to Modeling and Simulation KU Learning Outcomes

1) Explain the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem. *[Understanding]*
2) Describe the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling. *[Understanding]*
3) Differentiate among the different types of simulations, including physical simulations, human-guided simulations, and virtual reality. *[Understanding]*

CN/Modeling and Simulation KU Learning Outcomes

1) Explain and give examples of the benefits of simulation and modeling in a range of important application areas. *[Understanding]*

CN/Processing KU Learning Outcomes

1) Describe or sketch a workflow for an existing computational process such as the creation of a graph based on experimental data. *[Understanding]*
2) Describe the process of converting an algorithm to machine-executable code. *[Understanding]*
3) Summarize the phases of software development and compare several common lifecycle models. *[Understanding]*
4) Describe underflow, overflow, round off, and truncation errors in data representations. *[Understanding]*
5) Describe potential cyber attacks involving digital data. *[Understanding]*

Discrete Structures (DS) Knowledge Area

DS/Sets, Relations, and Functions KU Learning Outcomes

1) Explain with examples the basic terminology of functions, relations, and sets. *[Understanding]*
2) Perform the operations associated with sets, functions, and relations. *[Applying]*
3) Compare practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context. *[Analyzing]*

DS/Basic Logic KU Learning Outcomes

1) Convert logical statements from informal language to propositional and predicate logic expressions. *[Understanding]*
2) Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems such as predicting the behavior of software or solving problems such as puzzles. *[Applying]*
3) Use the rules of inference to construct proofs in propositional and predicate logic. *[Applying]*

4) Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g. program correctness), database queries, and algorithms. *[Understanding]*
5) Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms. *[Applying]*
6) Describe the strengths and limitations of propositional and predicate logic. *[Understanding]*

DS/Proof Techniques KU Learning Outcomes

1) Outline the basic structure of each proof technique (direct proof, proof by contradiction, and induction) described in this unit. *[Analyzing]*
2) Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument. *[Applying]*
3) Deduce the best type of proof for a given problem. *[Analyzing]*
4) Explain the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures. *[Understanding]*
5) Explain the relationship between weak and strong induction and give examples of the appropriate use of each. *[Understanding]*

DS/Basics of Counting KU Learning Outcomes

1) Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions. *[Applying]*
2) Apply the pigeonhole principle in the context of a formal proof. *[Applying]*
3) Calculate permutations and combinations of a set, and interpret the meaning in the context of the particular application. *[Applying]*
4) Compare real-world applications to appropriate counting formalisms. *[Analyzing]*
5) Solve a variety of basic recurrence relations. *[Applying]*
6) Analyze a problem to determine underlying recurrence relations. *[Analyzing]*
7) Perform computations involving modular arithmetic. *[Applying]*

DS/Graphs and Trees KU Learning Outcomes

1) Illustrate the basic terminology of graph theory including properties and special cases for each type of graph/tree. *[Applying]*
2) Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. *[Understanding]*
3) Solve a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system. *[Applying]*
4) Implement and use balanced trees and B-trees. *[Applying]*
5) Implement graph algorithms, i.e., graph search, union-find, minimum spanning trees, and shortest paths. *[Applying]*
6) Demonstrate how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting. *[Understanding]*
7) Describe binary search trees and AVL trees. *[Understanding]*

8) Explain complexity in the ideal and in the worst case scenario for both implementations. *[Understanding]*

DS/Discrete Probabilities KU Learning Outcomes

1) Calculate probabilities of events and expectations of random variables for elementary problems. *[Applying]*
2) Differentiate between dependent and independent events. *[Understanding]*
3) Identify a case of the binomial distribution and compute a probability using that distribution. *[Remembering]*
4) Apply Bayes theorem to determine conditional probabilities in a problem. [Applying]
5) Apply the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing. *[Applying]*

Graphics and Visualization (GV) Knowledge Area

GV/Fundamental Concepts KU Learning Outcomes

1) Explain the progression of Dimension and Coordinate System. *[Understanding]*
2) Describe common uses of digital presentation to human senses. (e.g., computer graphics, sound, haptic devices). *[Understanding]*
3) Explain in general terms how analog signals can be reasonably represented by discrete samples, for example, how images can be represented by pixels. *[Understanding]*
4) Explain how the limits of human perception affect choices about the digital representations. *[Understanding]*
5) Diagram a simple user interface using a standard API. *[Applying]*
6) Describe the differences in usage of file-types. (eg. lossy and lossless image compression techniques). *[Understanding]*
7) Describe color models and their use in graphics display devices. *[Understanding]*
8) Differentiate vector and raster rendering, resolution dependence and independence. *[Understanding]*
9) Compare techniques for developing a motion sequence. (discrete frames vs tweening) *[Analyzing]*
10) Analyze visualization techniques based on the problem needs. [Analyzing]
11) Use a multi-step Design Process to determine the problem, assess the audience and solve the problem. *[Applying]*
12) Diagram a program that has dynamic flexibility to multiple platforms (fluid layout or responsive design). *[Applying]*
13) Describe the conversion between different types of media. *[Understanding]*
14) Compare media delivery platforms identifying both security vulnerabilities and strengths. *[Analyzing]*
15) Use good file management techniques for backup and disaster planning. *[Applying]*
16) Use SWOT (Strength, Weaknesses, Threats, Opportunities) and other analysis methods to help identify needs. *[Applying]*

Human-Computer Interaction (HCI) Knowledge Area

HCI/Foundations KU Learning Outcomes

1) Discuss why human-centered software development is important. *[Understanding]*
2) Use a conceptual vocabulary for analyzing human interaction with software: affordance, conceptual model, feedback, and so forth. *[Applying]*
3) Illustrate a user-centered design process that explicitly takes account of the fact that the user is not like the developer or their acquaintances. *[Applying]*
4) Implement a simple usability test for an existing software application. *[Applying]*
5) Describe the first principles of security for software design. *[Understanding]*
6) Describe why each principle of secure software design is important to security and how to incorporate each principle into software design. *[Understanding]*
7) Explain the interaction between security and system usability and the importance for minimizing the effects of security mechanisms. *[Understanding]*

HCI/Designing Interaction KU Learning Outcomes

1) Illustrate a simple application, together with help and documentation, that supports a graphical user interface. *[Applying]*
2) Discuss at least one national or international user interface design standard. *[Understanding]*
3) Analyze and document the interface needs of an identified user group. *[Analyzing]*
4) Describe the issues of trust in interface design with an example of a high and low trust system. *[Understanding]*

Information Assurance and Security (IAS) Knowledge Area

IAS/Foundational Concepts in Security KU Learning Outcomes

1) Recognize the importance of security as a continuous process and its balancing nature between protection mechanisms and availability of data and information. *[Remembering]*
2) Differentiate between Information Security and Information Assurance. *[Understanding]*
3) Describe the concepts of risk, threats, vulnerabilities, attack vectors, and exploits (including the fact that there is no such thing as perfect security). *[Understanding]*
4) Explain the importance of security controls and countermeasures to minimize security risk and exposure. *[Understanding]*
5) Analyze the tradeoffs of balancing security properties (Confidentiality, Integrity, Availability, as well as Authentication, Authorization, Access, Authenticity, Non-Repudiation, Privacy). *[Analyzing]*
6) Explain the concepts of trust and trustworthiness. *[Understanding]*
7) Describe important ethical issues to consider in security. *[Understanding]*
8) Investigate risks to privacy and anonymity in technology. *[Applying]*
9) Apply cybersecurity principles to a changing landscape. *[Applying]*
10) Discuss the benefits of having multiple layers of defenses. *[Understanding]*

IAS/Principles of Secure Design KU Learning Outcomes

1) Describe the principle of least privilege and isolation as applied to system design. *[Understanding]*
2) Summarize the principles of fail-safe and deny-by-default. *[Understanding]*
3) Discuss the implications of relying on open design or the secrecy of design for security. *[Understanding]*
4) Explain the goals of end-to-end data security*. [Understanding]*
5) Discuss the benefits of having multiple layers of defenses (Defense In Depth). *[Understanding]*
6) For each stage in the lifecycle of a product, investigate what security considerations should be evaluated*. [Applying]*
7) Recognize the tradeoffs associated with designing security into a product. *[Remembering]*

IAS/Defensive Programming KU Learning Outcomes

1) Implement input validation and data sanitization in applications as necessary considering adversarial control of the input channel. *[Applying]*
2) Explain the tradeoffs of developing a program in a type-safe language. *[Understanding]*
3) Implement programs that properly handle exceptions and error conditions. *[Applying]*
4) Recognize the need to update software to fix security vulnerabilities. *[Remembering]*

IAS/Threats and Attacks KU Learning Outcomes

1) Identify likely attack types against standalone and networked software systems. *[Remembering]*
2) Describe risks to privacy and anonymity in information systems. *[Understanding]*
3) Discuss the key principles, such as membership and trust, of social engineering. *[Understanding]*

IAS/Cryptography KU Learning Outcomes

1) Explain the purpose of cryptography and how it is used to secure data. *[Understanding]*
2) Define key terms in cryptology*. [Remembering]*
3) Describe basic methods for transforming plaintext into ciphertext. *[Understanding]*
4) Explain the difference between symmetric and asymmetric encryption and how they are collectively used to secure digital communications and e-commerce transactions. *[Understanding]*

IAS/Web Security KU Learning Outcomes

1) Explain browser and web security model concepts including same-origin policy, web sessions, and secure communication channels. *[Understanding]*
2) Investigate common vulnerabilities and attacks in web applications and the coding strategies that are used to mitigate them. *[Applying]*

IAS/Secure Software Engineering KU Learning Outcomes

1) Write software requirements that include basic security specifications. *[Applying]*
2) Implement a plan to test the security modules in software. *[Applying]*
3) Investigate security vulnerabilities in a software at the requirement and design phases of the software development life cycle. *[Applying]*

Information Management (IM) Knowledge Area

IM/Information Management Concepts KU Learning Outcomes

1) Describe how humans gain access to information and data to support their needs. *[Understanding]*
2) Describe the advantages and disadvantages of central organizational control over data. *[Understanding]*
3) Summarize the careers/roles associated with information management. *[Understanding]*
4) Differentiate information with data. *[Understanding]*
5) Describe potential system attacks and the actors that might perform them. *[Understanding]*
6) Describe contingency plans for various size organizations to include: business continuity, disaster recovery and incident response. *[Understanding]*
7) Describe specific plans to secure data and information. *[Understanding]*
8) Discuss vulnerabilities and failure scenarios in common forms of information systems. *[Understanding]*

IM/Database Systems KU Learning Outcomes

1) Explain the characteristics that distinguish the database approach from the approach of programming with data files*. [Understanding]*
2) Describe the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor*. [Understanding]*
3) Summarize the basic goals, functions, and models of database systems. [Understanding]
4) Describe the components of a database system and give examples of their use. *[Understanding]*
5) Describe the roles of major DBMS functions in a database system. *[Understanding]*
6) Explain the concept of data independence and its importance in a database system. *[Understanding]*
7) Use a declarative query language to elicit information from a database. *[Applying]*
8) Describe common security concerns in database management systems. *[Understanding]*
9) Apply security principles to the design and development of database systems and database structures. *[Applying]*

IM/Data Modeling KU Learning Outcomes

1) Contrast appropriate data models, including internal structures, for different types of data. *[Analyzing]*
2) Describe concepts in modeling notation (e.g., Entity-Relation Diagrams or UML) and how they would be used. *[Understanding]*

3) Explain the fundamental terminology used in the relational data model. *[Understanding]*
4) Describe the basic principles of the relational data model. *[Understanding]*
5) Apply the modeling concepts and notation of the relational data model. *[Applying]*
6) Diagram a relational data model for a given scenario. [Understanding]
7) Describe the basic concepts of the OO model. *[Understanding]*
8) Describe the differences between relational data models and other models such as semi-structured or flexible schema (e.g., JSON, NoSQL). *[Understanding]*
9) Describe relevant security and privacy issues given a system and data management structure. *[Understanding]*

Networking and Communications (NC) Knowledge Area

NC/Introduction KU Learning Outcomes

1) Explain the basic structure of the Internet. *[Understanding]*
2) Define basic network terminology. *[Remembering]*
3) Describe the layered structure of a typical networked architecture. *[Understanding]*
4) Diagram the layers of the OSI model. *[Applying]*

NC/Networked Applications KU Learning Outcomes

1) List the differences and the relationships between names and addresses in a network. *[Remembering]*
2) Define the principles behind naming schemes and resource location. *[Remembering]*
3) Implement a simple client-server socket-based application. *[Applying]*

NC/Routing and Forwarding KU Learning Outcomes

1) Describe how packets are forwarded in an IP network. *[Understanding]*
2) Differentiate between routing and switching. *[Understanding]*

NC/Local Area Networks KU Learning Outcomes

1) Describe how frames are forwarded in a Local Area Network. *[Understanding]*
2) Describe the resources and services that Local Area Networks support. *[Understanding]*

NC/Mobility KU Learning Outcomes

1) Describe the organization of a wireless network. *[Understanding]*
2) Describe how wireless networks support mobile users. *[Understanding]*

Operating Systems (OS) Knowledge Area

OS/Overview of Operating Systems KU Learning Outcomes

1) Explain major objectives, functions and concepts of modern operating systems. *[Understanding]*

2) Describe key features of a contemporary operating system, such as scripting, user interfaces, and updates, with respect to convenience, efficiency, and the ability to evolve. *[Understanding]*
3) Differentiate between prevailing types of operating systems, such as networked, mobile, real-time, and distributed. *[Understanding]*
4) Discuss potential threats to operating systems and the security features designed to guard against them. *[Understanding]*

OS/Operating System Principles KU Learning Outcomes

1) Describe the value of APIs and middleware. *[Understanding]*
2) Describe how computing resources are used by application software and managed by system software. *[Understanding]*
3) Define a device list and a driver I/O queue. *[Remembering]*

OS/Concurrency KU Learning Outcomes

1) Describe the need for concurrency within the framework of an operating system. *[Understanding]*

OS/Memory Management KU Learning Outcomes

1) Describe the principles of memory management, including the function of and need for cache memory. *[Understanding]*
2) Describe the principles of virtual memory as applied to caching and paging. *[Understanding]*
3) Define the concept of thrashing. *[Remembering]*

OS/Security and Protection KU Learning Outcomes

1) Explain the need for protection and security in an OS. *[Understanding]*
2) Summarize the features of an operating system used to provide protection and security. *[Understanding]*
3) Explain the mechanisms available in an OS to control access to resources. *[Understanding]*

OS/Virtual Machines KU Learning Outcomes

1) Explain the concept of virtualization and how it is realized in hardware and software. *[Understanding]*

OS/Device Management KU Learning Outcomes

1) Explain the relationship between the physical hardware and the virtual devices maintained by the operating system. *[Understanding]*

Programming Languages (PL) Knowledge Area

PL/Object-Oriented Programming KU Learning Outcomes

1) Implement a simple class hierarchy, including superclasses and subclasses, using encapsulation, abstraction, inheritance, and polymorphism. *[Applying]*
2) Use control flow in a program using dynamic dispatch. *[Applying]*
3) Develop secure GUI applications using modern GUI development libraries and tools. *[Creating]*
4) Use private and protected methods to secure class data and to demonstrate encapsulation. *[Applying]*
5) Apply fundamental security principles and strategies to software development to inhibit attacks. *[Applying]*
6) Illustrate the secure development lifecycle. *[Applying]*
7) Assess secure coding by checking parameters based on certain data restrictions. *[Applying]*
8) Describe the tenets of OOP: encapsulation, abstraction, inheritance, and polymorphism and how they impact security. *[Understanding]*
9) Describe the characteristics of static, stack, and heap allocation. *[Understanding]*
10) Describe the state of the call stack when calling non-recursive and recursive subroutines. *[Understanding]*
11) Explain the difference between method definition and method calling. *[Understanding]*
12) Describe confidentiality, integrity, and availability and the impact each has on security. *[Understanding]*

PL/Functional Programming KU Learning Outcomes

1) Write basic algorithms that avoid assigning to mutable state or considering reference equality. *[Applying]*
2) Compare and contrast the procedural/functional approach and the object-oriented approach. *[Analyzing]*
3) Write useful functions that take and return other functions. *[Applying]*

PL/Event Driven and Reactive Programming KU Learning Outcomes

1) Write event handlers for use in reactive systems, such as GUIs. *[Applying]*
2) Explain why an event-driven programming style is natural in domains where programs react to external events. *[Understanding]*
3) Describe an interactive system in terms of a model, a view, and a controller. *[Understanding]*

4) Describe how event-driven GUI applications are structured when guarding against injection-based attacks*. [Understanding]*

PL/Basic Type Systems KU Learning Outcomes

1) Describe examples of program errors detected by a type system. *[Understanding]*
2) For multiple programming languages, identify program properties checked statically and program properties checked dynamically. *[Remembering]*

3) Write an example program that does not type-check in a particular language and yet would have no error if run. *[Applying]*
4) Use types and type-error messages to write and debug programs. *[Applying]*
5) Describe how object-oriented languages such as Java is strong-type language when define a data type in compile time. *[Understanding]*
6) Explain why you might choose to develop a program in a type-safe language in contrast to an unsafe programming language. *[Understanding]*

Software Development Fundamentals (SDF) Knowledge Area

SDF/Algorithms and Design KU Learning Outcomes

1) Discuss the importance of algorithms in the problem-solving process. *[Understanding]*
2) Discuss how a problem may be solved by multiple algorithms, each with different properties. *[Understanding]*
3) Use a programming language to implement algorithms designed to solve simple problems. *[Applying]*
4) Implement, test, and debug simple recursive functions and procedures. *[Applying]*
5) Apply the techniques of decomposition to break a program into smaller pieces. *[Applying]*
6) Describe the data components and behaviors of multiple abstract data types. *[Understanding]*
7) Write simple programs which use abstract data types (ADTs). *[Applying]*
8) Identify the relative strengths and weaknesses among multiple designs or implementations for a problem. *[Remembering]*
9) Demonstrate brute-force algorithms vs divide and conquer algorithms to accomplish security objectives: e.g., attempt to breaking a password. *[Understanding]*
10) Explain the difference and complexity between iterative-based and recursive-based implementations. *[Understanding]*
11) Choose whether a recursive or iterative solution is most appropriate for a problem. *[Evaluating]*

SDF/Fundamental Programming Concepts KU Learning Outcomes

1) Describe the characteristics of secure programming. *[Understanding]*
2) Discuss the importance of usability in security mechanism design. *[Understanding]*
3) Summarize the principle of fail-safe and deny-by-default. *[Understanding]*
4) Produce software components that satisfy their functional requirements without introducing vulnerabilities. *[Applying]*
5) Write code which uses defensive programming methods, such as input validation, type checking and buffer overflow. *[Applying]*
6) Examine security objectives by identifying bad input from the user according to the program design. *[Analyzing]*

SDF/Fundamental Data Structures KU Learning Outcomes

1) Describe common applications for each of the following data structures: stack, queue, priority queue, set, and map. *[Understanding]*

2) Write programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps. *[Applying]*
3) Compare alternative implementations of data structures with respect to performance. *[Analyzing]*
4) Describe how references allow for objects to be accessed in multiple ways. *[Understanding]*
5) Choose the appropriate data structure to solve a problem. *[Evaluating]*

SDF/Development Methods KU Learning Outcomes

1) Describe common coding errors that introduce security vulnerabilities. *[Understanding]*
2) Perform a code review on a program component. *[Applying]*
3) Describe opportunities within given program components for simple refactoring. *[Understanding]*
4) Describe contract programming and the role of preconditions, postconditions, and invariants. *[Understanding]*
5) Apply a variety of strategies to test and debug simple programs. *[Applying]*
6) Use an IDE to create, execute, and debug programs. *[Applying]*
7) Use standard libraries for a given programming language to create, execute, and debug programs. *[Applying]*
8) Apply consistent documentation and program style standards. *[Applying]*

Software Engineering (SE) Knowledge Area

SE/Software Processes KU Learning Outcomes

1) Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems. *[Understanding]*
2) Describe the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile). *[Understanding]*
    3) Describe the different practices that are key components of various process models. *[Understanding]*
    4) Differentiate among the phases of software development. [Understanding]
5) Describe how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes. *[Understanding]*
6) Describe the relative advantages and disadvantages among several major process models (e.g., multi-level security, waterfall with security, comprehensive lightweight application security process (CLASP), Extreme Programming, Aspect-oriented programming). *[Understanding]*

SE/Software Project Management KU Learning Outcomes

1) Discuss common behaviors that contribute to the effective functioning of a team. *[Understanding]*
    2) Describe necessary roles in a software development team. *[Understanding]*
    3) Describe the sources, hazards, and potential benefits of team conflict.

[Understanding]

4) Apply a conflict resolution strategy in a team setting. *[Applying]*
5) Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required. *[Applying]*
6) Describe different categories of risk in software systems. *[Understanding]*

7) Discuss the need to update software to fix security vulnerabilities and the life cycle management of the fix. *[Understanding]*

SE//Tools and Environments KU Learning Outcomes

1) Describe the difference between centralized and distributed software configuration management. *[Understanding]*
2) Describe how version control can be used to help with software release management. *[Understanding]*
3) Explain configuration items and use a source code control tool in a small team-based project. *[Understanding]*
4) Describe how available static and dynamic test tools can be integrated into the software development environment. *[Understanding]*
5) Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing. *[Understanding]*
6) Demonstrate the capability to use software tools in support of the development of a software product of medium size. *[Understanding]*
7) Use a modern IDE and debugger for security-minded debugging and testing. *[Applying]*
8) Explain the risks with misusing interfaces with third-party code and how to correctly use third-party code. *[Understanding]*
9) Use static and dynamic tools to identify programming faults. *[Applying]*

SE/Requirements Engineering KU Learning Outcomes

1) Describe the key components of a use case or similar description of some behavior that is required for a system. *[Understanding]*
2) Describe how the requirements engineering process supports the elicitation and validation of behavioral requirements. *[Understanding]*
3) Interpret a given requirements model for a simple software system. *[Understanding]*
4) Write system requirements from a client concept/specification that incorporate threat models. *[Applying]*
5) Write user narratives in preparation for building a piece of software. *[Applying]*
6) Describe important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing. *[Understanding]*
7) Describe the concepts of risk, threats, vulnerabilities and attack vectors (including the fact that there is no such thing as perfect security). *[Understanding]*
8) Explain the concept of trust and trustworthiness. *[Understanding]*
9) Explain the concepts of authentication, authorization, access control. *[Understanding]*

SE/Software Design KU Learning Outcomes

1) Describe different system design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures. *[Understanding]*
2) Analyze an existing software implementation and make suggestions to improve security in its design. *[Analyzing]*
3) Implement security improvements in an existing software implementation. *[Applying]*
4) Describe standard components for security operations, and explain the benefits of their use instead of re- inventing fundamentals operations. *[Understanding]*
5) Describe the concept of mediation and the principle of complete mediation. *[Understanding]*
6) Describe the cost and tradeoffs associated with designing security into a product. *[Understanding]*
7) Describe the requirements for integrating security into the software development lifecycle. *[Understanding]*
8) Explain the concept of trusted computing including trusted computing base and attack surface and the principle of minimizing trusted. *[Understanding]*

SE/Software Construction KU Learning Outcomes

1) Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness. *[Understanding]*
2) Classify robust code using exception handling mechanisms. *[Understanding]*
3) Describe secure coding and defensive coding practices. *[Understanding]*
4) Analyze a retired system for actionable attack information. *[Analyzing]*
5) Describe the process of analyzing and implementing changes to code base developed for a specific project. *[Understanding]*
6) Edit a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions, and test to insure the components are resilient to input and run-time errors. *[Applying]*
7) Diagram use cases. *[Applying]*
8) Examine the vulnerabilities in a given development process to an insider inserting undetected backdoor insertion. *[Analyzing]*
9) Use a defined coding standard in a small software project. *[Applying]*

SE/Software Verification and Validation KU Learning Outcomes

1) Distinguish between program validation and verification. *[Analyzing]*
2) Describe the role that tools can play in the validation of software. *[Understanding]*
3) Describe among the different types and levels of testing (unit, integration, systems, and acceptance). *[Understanding]*
4) Describe techniques to identify and select optimal and significant test cases for integration, regression and system testing. *[Understanding]*
5) Use a defect tracking tool to manage software defects in a small software project. *[Applying]*

6) Discuss the limitations of testing in a particular domain. *[Understanding]*

7) Describe input validation and data sanitization including how code is tested for input handling errors and the impact this has on security. *[Understanding]*

SE/Software Evolution KU Learning Outcomes

1) Identify the principal issues associated with software evolution and explain their impact on the software lifecycle. *[Remembering]*
2) Use refactoring in the process of modifying a software component. *[Applying]*
3) Discuss the challenges of evolving systems in a changing environment. *[Understanding]*
4) Outline the process of regression testing and its role in release management. *[Analyzing]*
5) Discuss the advantages and disadvantages of different types of software reuse. *[Understanding]*
6) Describe software development best practices for minimizing vulnerabilities in programming code. *[Understanding]*

SE/Software Reliability KU Learning Outcomes

1) Explain the problems that exist in achieving very high levels of reliability. *[Understanding]*
2) Describe how software reliability contributes to system reliability. *[Understanding]*
3) Identify ways to apply redundancy to achieve fault tolerance for a medium-sized application. *[Remembering]*
4) Analyze OOP design patterns and explain how they impact reliability and security. *[Analyzing]*
5) List approaches to minimizing faults that can be applied at each stage of the software lifecycle. *[Remembering]*

System Fundamentals (SF) Knowledge Area

SF/Computational Paradigms KU Learning Outcomes

1) List commonly encountered patterns of how computations are organized. *[Remembering]*
2) Identify the basic building blocks of computers and their role in the historical development of computer architecture. *[Remembering]*
3) Discuss the differences between single thread and multiple thread, single server and multiple server models. *[Understanding]*
4) Illustrate performance of simple sequential and parallel versions of a program with different problem sizes. *[Applying]*
5) Recognize security implications related to emerging computational paradigms. *[Remembering]*

SF/Cross-Layer Communications KU Learning Outcomes

1) Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers. *[Understanding]*
2) Implement a simple program using methods of layering, error detection and recovery, and reflection of error status across layers. *[Applying]*
3) Investigate bugs in a layered program using tools for program tracing, single stepping, and debugging. *[Applying]*

Social Issues and Professional Practice (SP) Knowledge Area

SP/Social Context KU Learning Outcomes

1) Describe positive and negative ways in which computer technology (networks, mobile computing, cloud computing) alters modes of social interaction at the personal level. *[Understanding]*
2) Interpret developers' assumptions and values embedded in hardware and software design, especially as they pertain to usability for diverse populations including under-represented populations and the disabled. *[Understanding]*
3) Interpret the social context of a given design and its implementation. *[Understanding]*
4) Describe the impact of the underrepresentation of diverse populations in the computing profession (e.g., industry culture, product diversity). *[Understanding]*

SP/Analytical Tools KU Learning Outcomes

1) Analyze stakeholder positions in a given situation. *[Analyzing]*
2) Discuss ethical/social tradeoffs in technical decisions. *[Understanding]*
3) Describe user responsibilities related to the handling of information in both personal and enterprise computing. *[Understanding]*
4) Describe potential cyber attacks and the actors that might perform them. *[Understanding*]

SP/Professional Ethics KU Learning Outcomes

1) Discuss various types of ethical issues and dilemmas in both personal and enterprise computing. *[Understanding]*
2) Explain recent and historical legislation related to digital privacy, unlawful access, and digital piracy, cyber defense, and computing ethics. *[Understanding]*
3) Analyze the impact of Acceptable Use Policies and Online Codes of Conduct on employee behavior and choices in the digital space. *[Analyzing]*
4) Summarize case studies related to ethics. *[Understanding]*

SP/Intellectual Property KU Learning Outcomes

1) Explain the terms intellectual property, fair-use, copyright, and plagiarism. *[Understanding]*
2) Discuss the key pieces of legislation related to fair-use, plagiarism, and intellectual property copyrights. *[Understanding]*
3) Summarize laws, both national and international, related to code patents and intellectual property copyrights. *[Understanding]*

SP/Privacy and Civil Liberties KU Learning Outcomes

1) Apply solutions to privacy threats in transactional databases and data warehouses. *[Applying]*
2) Discuss the role of data collection in the implementation of pervasive surveillance systems (e.g., RFID, face recognition, mobile computing). *[Understanding]*
3) Investigate the impact of technological solutions to privacy problems. *[Applying]*

SP/Professional Communication KU Learning Outcomes

1) Demonstrate competency in oral, written, and visual communication in the computing profession. *[Understanding]*
2) Distinguish between verbal and nonverbal communication. *[Analyzing]*
3) Demonstrate a broad grasp of communication theories as they apply to communication in the world of technology. *[Understanding]*

SP/Sustainability KU Learning Outcomes

1) Describe the economic, social, and environmental impacts of computing. *[Understanding]*
2) Discuss strategies used to assess and lessen the carbon footprint of materials and equipment used in computing. *[Understanding]*
3) Summarize case studies related to sustainable computing efforts. *[Understanding]*

SP/Security Policies, Laws and Computer Crime KU Learning Outcomes

1) List examples of computer crimes and social engineering incidents with societal impact. *[Remembering]*
2) Interpret laws that apply to computer crimes. *[Understanding]*
3) Describe the motivation and ramifications of cyber terrorism and criminal hacking. *[Understanding]*
4) Examine the ethical and legal issues surrounding the misuse of access and various breaches in security. *[Analyzing]*
5) Write a company-wide policy, which includes procedures for managing passwords, avoiding social engineering attacks, and employee monitoring. *[Applying]*

Assessment Metrics

This section presents all the student learning outcomes in the abridged version of the CS2013 Body of Knowledge as sample assessment rubrics, organized by Knowledge Area (KA) and subsequently by Knowledge Unit (KU). The learning outcomes are expressed using actions verbs from the cognitive domain of Bloom's Revised Taxonomy (http://ccecc.acm.org/assessment/blooms). The ACM CCECC uses a structured assessment rubric comprised of three tiers: "emerging", "developed", and "highly developed." Typically as the level of student achievement progresses from "emerging" to "highly developed", the level of Bloom's verbs also increases from the lower order thinking skills (LOTS) to the higher order thinking skills (HOTS).

Cybersecurity-related learning outcomes, except for the Information Assurance and Security (IAS) KA, are easily identified in **red font**.

Learning Outcome	Assessment Rubric		
AL. Algorithms and Complexity KA	**Emerging**	**Developed**	**Highly Developed**
AL/Basic Analysis Knowledge Unit			
1. Explain the differences among best, average, and worst case behaviors of an algorithm.	Label best, average, and worst bounds in a plot of a common function.	Explain the differences among best, average, and worst case behaviors of an algorithm.	Illustrate the differences among best, average, and worst case behaviors of an algorithm.
2. Calculate informally the time and space complexity of simple algorithms.	Discuss the time and space complexity of simple algorithms.	Calculate informally the time and space complexity of simple algorithms.	Deduce the complexity class of an algorithm by identifying loop nesting (for linear and above complexity classes) or repeatedly dividing by a constant (logarithmic).
3. Explain Big O notation.	State the formal definition of Big O notation.	Paraphrase the mathematical definition of Big O.	Illustrate Big O notation.
4. Classify algorithms based upon their Big O notation.	Recall Big O notation of common algorithms.	Explain, in words, why an algorithm belongs to a given complexity class.	Investigate the complexity class of a newly presented algorithm.
5. Contrast standard complexity classes.	Illustrate a few of the standard complexity classes.	Contrast standard complexity classes.	Evaluate standard complexity classes to classify algorithms into "efficient" and "inefficient".

6. Execute algorithms on input of various sizes and compare performance.	Discuss algorithms on input of various sizes and compare performance.	Execute algorithms on input of various sizes and compare performance.	Compare algorithms on input of various sizes and compare performance.
7. Analyze time and space trade-offs in algorithms.	Describe time and space trade-offs in algorithms.	Analyze time and space trade-offs in algorithms.	Evaluate time and space trade-offs in algorithms.
8. Explain the importance of writing secure and robust programs.	Recall the definitions of security and robustness in programs.	Explain the importance of writing secure and robust programs.	Exemplify a technique to make a program more secure or more robust.
AL/Algorithmic Strategies Knowledge Unit			
1. List examples of direct and indirect information flows.	Recognize conditional statements and repetitive structures that can manipulate the information flow.	List examples of direct and indirect information flows.	Implement a given information flow and produce an output based on different inputs.
2. Identify a variety of data structures including stacks, queues, priority queues, references, linked structures, resizable arrays, trees, graphs, and hash tables.	List advantages and disadvantages for each data structure.	Identify a variety of data structures including stacks, queues, priority queues, references, linked structures, resizable arrays, trees, graphs, and hash tables.	Implement each data structure and compare their complexities in time performance based on data input and structure of the data.
3. Demonstrate how to access certain positions of an array.	Recall the syntax for array access; recall how to determine the length of an array.	Describe how to access the midpoint element of an array and the last element of an array, given the length of an array.	Implement a method that given an array as an argument will print alternatively the elements of the array from the front and the end until find the midpoint.
4. Demonstrate which data model (list, set, hash table, tree, or graph) is appropriate for solving a problem.	List advantages and disadvantages of each data model for a given problem.	Demonstrate which data model (list, set, hash table, tree, or graph) is appropriate for solving a problem.	Test a data model with different data formats (include duplicates, ascending and descending order, and random order).
5. Compare and contrast approaches for resolving collisions in hash tables, e.g., linear probing, quadratic probing, double hashing, rehashing, and	List collision resolution schemes.	Compare and contrast approaches for resolving collisions in hash tables, e.g., linear probing,	Evlauate code that uses collision resolution to perform key-value pair insertions and

chaining.		quadratic probing, double hashing, rehashing, and chaining.	retrievals in a hash table.
6. Assess secure programming by writing a program that checks boundaries in an array as an example of a buffer over/underflow.	Explain underflow and overflow.	Assess secure programming by writing a program that checks boundaries in an array as an example of a buffer over/underflow.	Create unit tests to verify the boundary checks.
7. Implement a recursive solution to a problem.	Identify a recursive pattern in a mathematical series (e.g., geometric or harmonic series).	Implement a recursive solution to a problem.	Examine a recursive solution to a problem.
AL/Fundamental Data Structures and Algorithms Knowledge Unit			
1. Implement basic numerical algorithms.	Explain convergence series as a usage of sigma or product.	Implement basic numerical algorithms.	Estimate the a value by using convergence series.
2. Implement simple search algorithms and explain the differences in their time complexities.	Name the linear and binary search .algorithms; state their time complexities.	Implement a method that takes a sorted array and performs binary search; explain why its time complexity is logarithmic.	Compare complexities of both implementations and define assumptions that make algorithms work as optimal.
3. Implement common quadratic a O(N log N) sorting algorithms.	Demonstrate the algorit mergesort by using a n array.	Implement the merge algorithm in mergesort; implement the partition algorithm in quicksort.	Implement mergesort a quicksort from scratch.
4. Describe the implementation of hash tables, including collision avoidance and resolution.	Explain the general idea of a hash table (i.e., exploit the constant-time access for an array). Explain the need for collision resolution strategies.	Describe the implementation of hash tables, including collision avoidance and resolution.	Given a hash function, create code that inserts a key-value pair into a hash table and code that retrieves a value given a key; both codes should take collision resolution into consideration.
5. Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing.	Recall the worst-case runtime and memory efficient of principal algorithms.	Discuss the runtime and memory efficiency of principal algorithms for sorting, searching,	Implement different collision resolutions for a hash table and compare the number

		and hashing.	of rehashing for each resolutions; discuss the time performance.
6. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data.	List other factors other than computational efficiency that ought to be considered when choosing an algorithm.	Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data.	Test an algorithm and consider computational efficiency factors and verify their impact in terms of time and space.
7. Explain how tree balance affects the efficiency of various binary search tree operations.	List differences between a balanced search tree and a degenerate tree.	Explain how tree balance affects the efficiency of various binary search tree operations.	Compare worst case scenarios of a balance-tree vs a regular BST.
8. Solve problems using fundamental graph algorithms, including depth-first and breadth-first search.	State the differences between depth-first and breadth-first search.	Solve problems using fundamental graph algorithms, including depth-first and breadth-first search.	Design algorithms to find strongly connected components based on the principles of depth-first search.
AL/Basic Automata, Computability and Complexity Knowledge Unit			
1. Use a regular expression to represent a specified language.	Explain what regular expressions are useful and their impact in a programming language.	Use a regular expression to represent a specified language.	Diagram a state diagram to express the recognition of a regular expression in a state machine.
2. Describe a deterministic finite state machine to accept a specified language.	List the possible states that a finite state machine can have to accept a specified language.	Describe a deterministic finite state machine to accept a specified language.	Design a deterministic finite state machine to accept a specified language.
3. Explain the role of random numbers in security, beyond just cryptography (e.g., password generation, randomized algorithms to avoid algorithmic denial of service attacks).	Diagram a state machine that will illustrate how a machine will match a valid/invalid password.	Explain the role of random numbers in security, beyond just cryptography (e.g. password generation, randomized algorithms to avoid algorithmic denial of service attacks).	Create a brute-force algorithm that attempts to match a password generated by random characters.
4. Discuss the concept of finite state machines.	Recognize the input, output, and the states a finite state machine has.	Discuss the concept of finite state machines.	Write a state machine that validates a given pattern based on a given rules.
5. Explain why the halting problem has no algorithmic solution.	Recognize problems that are considered to	Explain why the halting problem has	Illustrate the proof of the halting problem by

	be undecidable.	no algorithmic solution.	demonstrating Turing machines.

Learning Outcome	Assessment Rubric		
AR. Architecture and Organization KA	**Emerging**	**Developed**	**Highly Developed**
AR/Machine Level Representation of Data Knowledge Unit			
1. Explain why everything is data, including instructions, in computers.	Recognize why everything is data, including instructions, in computers.	Explain why everything is data, including instructions, in computers.	Investigate why everything is data, including instructions, in computers.
2. Explain reasons for using alternative formats to represent numerical data.	Recognize the reasons for using alternative formats to represent numerical data.	Explain reasons for using alternative formats to represent numerical data.	Apply alternative formats to represent numerical data.
3. Explain how fixed-length number representations affect accuracy and precision.	Define fixed-length number representations.	Explain how fixed-length number representations affect accuracy and precision.	Illustrate how fixed-length number representations affect accuracy and precision.
4. Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays.	Define the internal representation of non-numeric data, such as characters, strings, records, and arrays.	Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays.	Illustrate internal representation of non-numeric data, such as characters, strings, records, and arrays.
5. Convert numerical data from one format to another, such as, negative integers into sign-magnitude and two's-complement representations.	Describe how to convert numerical data from one format to another.	Convert numerical data from one format to another, such as, negative integers into sign-magnitude and two's-complement representations.	Compare and contrast different methods for converting numerical data from one format to another.
AR/Assembly Level Machine Organization Knowledge Unit			
1. Explain the organization of the classical von Neumann machine and its major functional units.	Define the organization of the classical von Neumann machine and its major functional units.	Explain the organization of the classical von Neumann machine and its major functional units.	Diagram the organization of the classical von Neumann machine and its major functional units.

2. Demonstrate how high-level language patterns map to assembly/machine language, including subroutine calls.	Recognize how high-level language patterns map to assembly/machine language, including subroutine calls.	Demonstrate how high-level language patterns map to assembly/machine language, including subroutine calls.	Diagram high-level language patterns map to assembly/machine language, including subroutine calls.
3. Explain the basic concepts of interrupts and I/O operations.	List basic concepts of interrupts and I/O operations.	Explain the basic concepts of interrupts and I/O operations.	Implement basic concepts of interrupts and I/O operations.
4. Write simple assembly language program segments.	Explain simple assembly language program segments.	Write simple assembly language program segments.	Write more complex assembly language program segments.
AR/Memory System Organization and Architecture Knowledge Unit			
1. Identify the main types of memory technology and their relative cost and performance.	Name the main types of memory technology (e.g., SRAM, DRAM, Flash, magnetic disk).	Identify the main types of memory technology (e.g., SRAM, DRAM, Flash, magnetic disk) and their relative cost and performance.	Differentiate the main types of memory technology (e.g., SRAM, DRAM, Flash, magnetic disk) and discuss their relative cost and performance.
2. Explain the effect of memory latency on running time.	Recognize the effect of memory latency on running time.	Explain the effect of memory latency on running time.	Calculate the effect of memory latency on running time.
3. Describe how the use of memory hierarchy is used to reduce the effective memory latency.	Recall how the use of memory hierarchy is used to reduce the effective memory latency.	Describe how the use of memory hierarchy (e.g., cache, virtual memory) is used to reduce the effective memory latency.	Investigate the use of memory hierarchy to reduce the effective memory latency.

Learning Outcome	**Assessment Rubric**		
CN. Computational Science KA	**Emerging**	**Developed**	**Highly Developed**
CN/Introduction to Modeling and Simulation Knowledge Unit			
1. Explain the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.	Define the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.	Explain the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.	Evaluate the concept of modeling and the use of abstraction that allows the use of a machine to solve a problem.

2. Describe the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling.	List the relationships between modeling and simulation	Describe the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling.	Illustrate the relationship between modeling and simulation, i.e., thinking of simulation as dynamic modeling.
3. Differentiate among the different types of simulations, including physical simulations, human-guided simulations, and virtual reality.	Identify the different types of simulations, including physical simulations, human-guided simulations, and virtual reality.	Differentiate among the different types of simulations, including physical simulations, human-guided simulations, and virtual reality.	Investigate the different types of simulations, including physical simulations, human-guided simulations, and virtual reality.
CN/Modeling and Simulation Knowledge Unit			
1. Explain and give examples of the benefits of simulation and modeling in a range of important application areas.	Identify the benefits of simulation and modeling in a range of important application areas.	Explain the benefits of simulation and modeling in a range of important application areas.	Investigate and show, using examples, the benefits of simulation and modeling in a range of important application areas.
CN/Processing Knowledge Unit			
1. Describe or sketch a workflow for an existing computational process such as the creation of a graph based on experimental data.	Identify a workflow for an existing computational process such as the creation of a graph based on experimental data.	Describe a workflow for an existing computational process such as the creation of a graph based on experimental data.	Diagram a workflow for an existing computational process such as the creation of a graph based on experimental data.
2. Describe the process of converting an algorithm to machine-executable code.	Define the process of converting an algorithm to machine-executable code.	Describe the process of converting an algorithm to machine-executable code.	Apply the process of converting an algorithm to machine-executable code.
3. Summarize the phases of software development and compare several common lifecycle models.	Identify the phases of software development and compare several common lifecycle models.	Summarize the phases of software development and compare several common lifecycle models.	Diagram the phases of software development and compare several common lifecycle models.
4. Describe underflow, overflow, round off, and truncation errors in data representations.	Explain how data is represented in a machine	Describe underflow, overflow, round off, and truncation errors in data representations.	Calculate results when there is an underflow, overflow, round off, and truncation errors.
5. Describe potential cyber attacks involving digital data.	Identify potential cyber attacks involving digital	Describe potential cyber attacks involving	Examine the risks of potential cyber attacks

	data.	digital data.	involving digital data.

Learning Outcome	Assessment Rubric		
DS. Discrete Structures KA	**Emerging**	**Developed**	**Highly Developed**
DS/Sets, Relations, and Functions Knowledge Unit			
1. Explain with examples the basic terminology of functions, relations, and sets.	Identify the defining features of functions, relations, and sets.	Explain with examples the basic terminology of functions, relations, and sets.	Use function, relations, and set terminology in a correct and meaningful way.
2. Perform the operations associated with sets, functions, and relations.	Describe the operations associated with sets, functions, and relations.	Perform the operations associated with sets, functions, and relations.	Compare the operations of sets, functions, and relations.
3. Compare practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context .	Implement a solution to a programming problem using a particular set, function, or relation model.	Compare practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.	Justify the choice of a particular set, function, or relation model.
DS/Basic Logic Knowledge Unit			
1. Convert logical statements from informal language to propositional and predicate logic expressions.	Recognize the relationship between logical statements from informal language and propositional and predicate logic expressions.	Convert logical statements from informal language to propositional and predicate logic expressions.	Produce propositional and predicate logic expressions from a given logical statement from an informal language.
2. Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems such as predicting the behavior of software or solving problems such as puzzles.	Describe the steps in formal logic proofs and/or informal logical reasoning to solve real problems.	Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems such as predicting the behavior of software or solving problems such as puzzles.	Compare different logic proofs and informal logical reasoning to determine correct methods to solve real problems.
3. Use the rules of inference to	Discuss the rules of	Use the rules of	Analyze the rules of infe

construct proofs in propositional and predicate logic.	inference to construct proofs in propositional and predicate logic.	inference to construct proofs in propositional and predicate logic.	construct proofs in prop predicate logic.
4. Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g. program correctness), database queries, and algorithms.	List ways that symbolic logic can be used to model real-life situations or applications.	Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g. program correctness), database queries, and algorithms.	Use symbolic logic to model real-life situations.
5. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms.	Demonstrate formal methods of symbolic propositional and predicate logic.	Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms.	Distinguish between formal methods of propositional and predicate logic to determine the most effective solutions to a given problem.
6. Describe the strengths and limitations of propositional and predicate logic.	List the strengths and limitations of propositional and predicate logic.	Describe the strengths and limitations of propositional and predicate logic.	Illustrate the strengths and limitations of propositional and predicate logic.
DS/Proof Techniques Knowledge Unit			
1. Outline the basic structure of each proof technique (direct proof, proof by contradiction, and induction) described in this unit.	Use the basic structure of each proof technique to solve a problem.	Outline the basic structure of each proof technique (direct proof, proof by contradiction, and induction).	Choose the most effective proof technique to solve a problem.
2. Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument.	Demonstrate each of the proof techniques by correctly constructing a sound argument.	Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument.	Use each of the proof techniques correctly in the construction of a sound argument.
3. Deduce the best type of proof for a given problem.	Compare the different proof methods.	Deduce the best type of proof for a given problem.	Construct a correct proof using the best method for a given problem.
4. Explain the parallels between	Identify the parallels	Explain the parallels	Illustrate the parallels

ideas of mathematical and/or structural induction to recursion and recursively defined structures.	between ideas of mathematical and/or structural induction to recursion and recursively defined structures.	between ideas of mathematical and/or structural induction to recursion and recursively defined structures.	between ideas of mathematical and/or structural induction to recursion and recursively defined structures.
5. Explain the relationship between weak and strong induction and give examples of the appropriate use of each.	Identify the relationship between weak and strong induction.	Explain the relationship between weak and strong induction and give examples of the appropriate use of each.	Solve problems using both weak and strong induction.
DS/Basics of Counting Knowledge Unit			
1. Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions.	Describe counting arguments.	Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions.	Outline counting arguments.
2. Apply the pigeonhole principle in the context of a formal proof.	Demonstrate the pigeonhole principle.	Apply the pigeonhole principle in the context of a formal proof.	Analyze the pigeonhole principle in the context of a formal proof.
3. Calculate permutations and combinations of a set, and interpret the meaning in the context of the particular application.	Explain the calculation of permutations and combinations of a set.	Calculate permutations and combinations of a set, and interpret the meaning in the context of the particular application.	Discriminate between computation of st permutations and combinations.
4. Compare real-world applications to appropriate counting formalisms.	Use counting formalisms to solve real-world applications.	Compare real-world applications to appropriate counting formalisms, such as determining the number of ways to arrange people around a table, subject to constraints on the seating arrangement, or the number of ways to determine certain hands in cards (e.g., a full house).	Choose appropriate counting formalisms to solve real-world applications.

5. Solve a variety of basic recurrence relations.	Demonstrate a variety of basic recurrence relations.	Solve a variety of basic recurrence relations.	Compare a variety of basic recurrence relations.
6. Analyze a problem to determine underlying recurrence relations.	Carry out a problem with an underlying recurrence relation.	Analyze a problem to determine underlying recurrence relations.	Evaluate a problem with an underlying recurrence relation.
7. Perform computations involving modular arithmetic.	Discuss computations involving modular arithmetic.	Perform computations involving modular arithmetic.	Examine computations involving modular arithmetic.
DS/Graphs and Trees Knowledge Unit			
1. Illustrate the basic terminology of graph theory including properties and special cases for each type of graph/tree.	Describe the basic terminology of graph theory.	Illustrate the basic terminology of graph theory including properties and special cases for each type of graph/tree.	Outline the basic terminology of graph theory.
2. Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees.	List the different traversal methods for trees and graphs	Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees.	Execute different traversal methods for trees and graphs.
3. Solve a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system.	Discuss a variety of real-world problems in computer science using appropriate forms of graphs and trees.	Solve a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system.	Distinguish between real-world problems solvable by using graphs and trees.
4. Implement and use balanced trees and B-trees.	Explain balanced trees and B-trees.	Implement and use balanced trees and B-trees.	Analyze the use of balanced trees and B-trees.
5. Implement graph algorithms, i.e., graph search, union-find, minimum spanning trees, and shortest paths.	Classify graph algorithms.	Implement graph algorithms, i.e., graph search, union-find, minimum spanning trees, and shortest paths.	Categorize different implementations of graph algorithms.
6. Demonstrate how concepts from graphs and trees appear in data structures, algorithms,	Identify how concepts from graphs and trees appear in data	Demonstrate how concepts from graphs and trees appear in	Implement data structures, algorithms, proof techniques, and

proof techniques (structural induction), and counting.	structures, algorithms, proof techniques, and counting.	data structures, algorithms, proof techniques (structural induction), and counting.	counting using graphs and trees.
7. Describe binary search trees and AVL trees.	Define binary search and AVL trees.	Describe binary search trees and AVL trees.	Apply binary search and AVL trees.
8. Explain complexity in the ideal and in the worst case scenario for both implementations.	State complexity in the ideal and in the worst case scenario for both implementations.	Explain complexity in the ideal and in the worst case scenario for both implementations.	Calculate complexity in the ideal and in the worst case scenario for both implementations.
DS/Discrete Probability Knowledge Unit			
1. Calculate probabilities of events and expectations of random variables for elementary problems.	Exemplify probabilities of events and expectations of random variables for elementary problems such as games of chance.	Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.	Examine probabilities of events and expectations of random variables for elementary problems such as games of chance.
2. Differentiate between dependent and independent events.	Identify dependent and independent events.	Differentiate between dependent and independent events.	Illustrate dependent and independent events.
3. Identify a case of the binomial distribution and compute a probability using that distribution.	Demonstrate a probability computed using a binomial distribution.	Identify a case of the binomial distribution and compute a probability using that distribution.	Analyze a probability computed using binomial distribution.
4. Apply Bayes theorem to determine conditional probabilities in a problem.	Explain Bayes theorem to determine conditional probabilities in a problem.	Apply Bayes theorem to determine conditional probabilities in a problem.	Outline Bayes theorem to determine conditional probabilities in a problem.
5. Apply the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing.	Discuss the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing.	Apply the tools of probability to solve problems such as the average case analysis of algorithms or analyzing hashing.	Analyze the tools of probability in solving problems such as the average case analysis of algorithms or analyzing hashing.

Learning Outcome	Assessment Rubric		
GV. Graphics and Visualization KA	**Emerging**	**Developed**	**Highly Developed**
GV/Fundamental Concepts Knowledge Unit			
1. Explain the progression of Dimension and Coordinate System.	Define dimensional and coordinate Systems.	Explain how to use dimensions and coordinate systems.	Compare transformation and changes in dimension and coordinate systems for 2D and 3D design.
2. Describe common uses of digital presentation to human senses. (e.g., computer graphics, sound, haptic devices).	List a variety of digital presentation media in relationship to human senses.	Describe the choices in multimodal media.	Choose appropriate digital presentation options based on project need.
3. Explain in general terms how analog signals can be reasonably represented by discrete samples, for example, how images can be represented by pixels.	List the advantages and disadvantages of display devices.	Explain file types and appropriate use.	Convert image types according to output choices.
4. Explain how the limits of human perception affect choices about the digital representations.	Define human vision and the limits in relationship to wavelength.	Explain limitations in choosing digital representations.	Evaluate the level of detail in making choices for project need.
5. Diagram a simple user interface using a standard API.	Define a project's technical options and tools during pre-production.	Diagram a simple user interface using a standard API.	Construct a simple user interface using a standard API.
6. Describe the differences in usage of file-types. (eg. lossy and lossless image compression techniques).	Define file types.	Describe the differences in the usage of file-types. (eg. lossy and lossless image compression techniques).	Debate the differences in the usage of file-types. (eg. lossy and lossless image compression techniques).
7. Describe color models and their use in graphics display devices.	Define color models.	Describe color models and their use in graphics display devices.	Assess color models and their use in graphics display devices.
8. Differentiate vector and raster rendering, resolution dependence and independence.	List the advantages and disadvantages to pixel vs. vector image structures.	Differentiate file types, resolution needs and appropriate use.	Analyze image types according to output choices.

9. Compare techniques for developing a motion sequence. (discrete frames vs tweening)	Explain how to create a basic motion sequence from discrete frames.	Compare techniques for developing a motion sequence. (discrete frames vs tweening)	Create a basic motion sequence from discrete frames.
10. Analyze visualization techniques based on the problem needs.	Select visualization techniques.	Analyze visualization techniques based on the problem needs.	Defend visualization techniques to problem/project.
11.Use a multi-step Design Process to determine the problem, assess the audience and solve the problem.	Define the steps in a production or design process.	Use a multi-step Design Process to determine the problem, assess the audience and solve the problem.	Develop a solution based on multi-step design process principles and produce a deliverable.
12. Diagram a program that has dynamic flexibility to multiple platforms (fluid layout or responsive design).	Define options in responsive design.	Diagram a program that has dynamic flexibility to multiple platforms (fluid layout or responsive design).	Create a multi-platform responsive design.
13. Describe the conversion between different types of media.	List media types.	Describe the conversion between different types of media.	Convert media types.
14. Compare media delivery platforms identifying both security vulnerabilities and strengths.	Identify security strengths and vulnerabilities in various media delivery platforms.	Compare media delivery platforms identifying both security vulnerabilities and strengths.	Evaluate media delivery platforms' security strengths and vulnerabilities.
15. Use good file management techniques for backup and disaster planning.	Differentiate file management options vulnerabilities and strengths.	Use good file management techniques for backup and disaster planning.	Test backup procedures and offsite storage usage as needed.
16.Use SWOT (Strength, Weaknesses, Threats, Opportunities) and other analysis methods to help identify needs.	Define SWOT Analysis.	Use SWOT Analysis.	Evaluate SWOT Analysis.

Learning Outcome	Assessment Rubric		
HCI. Human Computer Interaction KA	**Emerging**	**Developed**	**Highly Developed**
HCI/Foundations Knowledge Unit			
1. Discuss why human-centered software development is important.	Define human-centered software.	Discuss why human-centered software development is important.	Defend the importance of Human Centered Software.
2. Use a conceptual vocabulary for analyzing human interaction with software: affordance, conceptual model, feedback, and so forth.	Define HCI vocabulary and demonstrate how it is used.	Use a conceptual vocabulary for analyzing human interaction with software: affordance, conceptual model, feedback, etc.	Create a HCI vocabulary that can be used to analyze HCI.
3.Illustrate a user-centered design process that explicitly takes account of the fact that the user is not like the developer or their acquaintances.	Identify user-centered design process.	Illustrate how user-centered design process is developed.	Assess the differences between the user and the developer.
4. Implement a simple usability test for an existing software application.	Define usability test.	Implement a simple usability test for an existing software application.	Hypothesize if the usability test created will be successful.
5. Describe the first principles of security for software design.	List the security principles related to software design.	Describe the first principles of security for software design.	Assess the importance of secure software design principles.
6. Describe why each principle of secure software design is important to security and how to incorporate each principle into software design.	List principles of secure software design and methods for incorporation.	Describe why each principle of secure software design is important to security and how to incorporate each principle into software design.	Analyze the impact of the failure to follow secure software design principles and incorporation methods.
7. Explain the interaction between security and system usability and the importance for minimizing the effects of security mechanisms.	Identify potential usability issues related security mechanisms.	Explain the interaction between security and system usability and the importance for minimizing the effects of security mechanisms.	Evaluate the usage of security mechanisms measured against usability affects.

HCI/Designing Interaction Knowledge Unit			
1. Illustrate a simple application, together with help and documentation, that supports a graphical user interface.	Design a simple application using HCI requirements.	Illustrate a simple application, together with help and documentation, that supports a graphical user interface.	Create a complete application that uses a graphical interface and includes technical documentation.
2. Discuss at least one national or international user interface design standard.	Identify national and international user interface design standards.	Discuss at least one national or international user interface design standard.	Justify at least one national and one international user interface design standard.
3. Analyze and document the interface needs of an identified user group.	List common user interface needs.	Analyze and document the interface needs of an identified user group.	Assess the appropriateness of design standards in meeting specified user interface needs.
4. Describe the issues of trust in interface design with an example of a high and low trust system.	List design elements that make a human-computer interface trustworthy.	Describe the issues of trust in interface design with an example of a high and low trust system.	Create both an interface design with high trust and another with low trust.

Learning Outcome	Assessment Rubric		
IAS. Information Assurance and Security KA	**Emerging**	**Developed**	**Highly Developed**
IAS/Foundational Concepts in Security Knowledge Unit			
1. Recognize the importance of security as a continuous process and its balancing nature between protection mechanisms and availability of data and information.	Recognize the importance of security.	Recognize the importance of security as a continuous process and its balancing nature between protection mechanisms and availability of data and information.	Explain the importance of security as a continuous process and its balancing nature between protection mechanisms and availability of data and information.
2. Differentiate between Information Security and Information Assurance.	Define Information Security and Information Assurance.	Differentiate between Information Security and Information Assurance.	Compare and contrast Information Security and Information Assurance

3. Describe the concepts of risk, threats, vulnerabilities, attack vectors, and exploits (including the fact that there is no such thing as perfect security).	Define the concepts of risk, threats, vulnerabilities, attack vectors, and exploits (including the fact that there is no such thing as perfect security).	Describe the concepts of risk, threats, vulnerabilities, attack vectors, and exploits (including the fact that there is no such thing as perfect security).	Apply the concepts of risk, threats, vulnerabilities, attack vectors, and exploits (including the fact that there is no such thing as perfect security) to a scenario.
4. Explain the importance of security controls and countermeasures to minimize security risk and exposure.	Recognize the importance of security controls and countermeasures to minimize security risk and exposure.	Explain the importance of security controls and countermeasures to minimize security risk and exposure.	Implement one or more security controls and countermeasures to minimize security risk and exposure.
5. Analyze the tradeoffs of balancing security properties (Confidentiality, Integrity, Availability, as well as Authentication, Authorization, Access, Authenticity, Non-Repudiation, Privacy).	Investigate the tradeoffs of balancing security properties (Confidentiality, Integrity, Availability, as well as Authentication, Authorization, Access, Authenticity, Non-Repudiation, Privacy).	Distinguish the tradeoffs of balancing security properties (Confidentiality, Integrity, Availability, as well as Authentication, Authorization, Access, Authenticity, Non-Repudiation, Privacy).	Evaluate the tradeoffs of balancing security properties (Confidentiality, Integrity, Availability, as well as Authentication, Authorization, Access, Authenticity, Non-Repudiation, Privacy).
6. Explain the concepts of trust and trustworthiness.	Define the concepts of trust and trustworthiness.	Explain the concepts of trust and trustworthiness.	Diagram trust relationships.
7. Describe important ethical issues to consider in security.	Identify important ethical issues to consider in security.	Describe important ethical issues to consider in security.	Investigate important ethical issues to consider in security.
8. Investigate risks to privacy and anonymity in technology.	Describe risks to privacy and anonymity in technology.	Investigate risks to privacy and anonymity in technology	Analyze risks to privacy and anonymity in technology.
9. Apply cybersecurity principles to a changing landscape.	Summarize cybersecurity principles and how they are affected by a changing landscape.	Apply cybersecurity principles to a changing landscape	Examine cybersecurity principles in a changing landscape.
10. Discuss the benefits of having multiple layers of defenses.	State the benefits of having multiple layers of defense.	Discuss the benefits of having multiple layers of defenses.	Implement multiple layers of defense.
IAS/Principles of Secure Design Knowledge Unit			
1. Describe the principle of least privilege and isolation as applied	Define the principle of least privilege and	Describe the principle of least privilege and	Given a scenario, interpret the concept of

to system design.	isolation.	isolation as applied to system design.	least privilege and isolation as applied to system design.
2. Summarize the principles of fail-safe and deny-by-default.	Define the principles of fail-safe and deny-by-default.	Summarize the principles of fail-safe and deny-by-default.	Differentiate the principles of fail-safe and deny-by-default.
3. Discuss the implications of relying on open design or the secrecy of design for security.	Recognize the implications of relying on open design or the secrecy of design for security.	Discuss the implications of relying on open design or the secrecy of design for security.	Illustrate the implications of relying on open design or the secrecy of design for security.
4. Explain the goals of end-to-end data security.	List some of the goals of end-to-end data security.	Explain the goals of end-to-end data security.	Illustrate using examples the goals of end-to-end data security.
5. Discuss the benefits of having multiple layers of defenses (Defense In Depth).	Recognize one or more of the benefits of having multiple layers of defenses (Defense In Depth).	Discuss the benefits of having multiple layers of defenses (Defense In Depth)	Implement multiple layers of defenses (Defense In Depth) for a given scenario
6. For each stage in the lifecycle of a product, investigate what security considerations should be evaluated.	For each stage in the lifecycle of a product, describe what security considerations should be evaluated.	For each stage in the lifecycle of a product, investigate what security considerations should be evaluated.	For each stage in the lifecycle of a product, analyze what security considerations should be evaluated.
7. Recognize the tradeoffs associated with designing security into a product.	Recognize some of the tradeoffs associated with designing security into a product.	Recognize the tradeoffs associated with designing security into a product.	Describe the tradeoffs associated with designing security into a product including cost and benefits.
IAS/Defensive Programming Knowledge Unit			
1. Implement input validation and data sanitization in applications as necessary considering adversarial control of the input channel.	Explain the importance of input validation and data sanitization in applications considering adversarial control of the input channel.	Implement input validation and data sanitization in applications as necessary considering adversarial control of the input channel.	Analyze the effectiveness of input validation and data sanitization implemented in applications considering adversarial control of the input channel.
2. Explain the tradeoffs of developing a program in a type-safe language.	List some of the tradeoffs of developing a program in a type-safe language.	Explain the tradeoffs of developing a program in a type-safe language.	Compare and contrast the tradeoffs of developing a program in a type-safe language.

3. Implement programs that properly handle exceptions and error conditions.	Explain the importance of writing programs that properly handle exceptions and error conditions.	Implement programs that properly handle exceptions and error conditions.	Analyze the implementation of exception handling in programs and error conditions.
4. Recognize the need to update software to fix security vulnerabilities.	Recognize the need to update software.	Recognize the need to update software to fix security vulnerabilities.	Demonstrate the need to update software to fix security vulnerabilities.
IAS/Threats and Attacks Knowledge Unit			
1. Identify likely attack types against standalone and networked software systems.	Identify some attack types against software systems.	Identify likely attack types against standalone and networked software systems.	Discuss likely attack types against standalone and networked software systems.
2. Describe risks to privacy and anonymity in information systems.	Name risks to privacy and anonymity in information systems.	Describe risks to privacy and anonymity in information systems.	Investigate risks to privacy and anonymity in information systems.
3. Discuss the key principles, such as membership and trust, of social engineering.	Recognize the key principles, such as membership and trust, of social engineering.	Discuss the key principles, such as membership and trust, of social engineering.	Illustrate the key principles, such as membership and trust, of social engineering.
IAS/Cryptography Knowledge Unit			
1. Explain the purpose of cryptography and how it is used to secure data.	State the purpose of cryptography.	Explain the purpose of cryptography and how it is used to secure data.	Implement cryptography to secure data.
2. Define key terms in cryptology.	List some key terms in cryptology.	Define key terms in cryptology, including cryptography, cryptanalysis, cipher, and cryptographic algorithm.	Explain key terms in cryptology, including cryptography, cryptanalysis, cipher, and cryptographic algorithm.
3. Describe basic methods for transforming plaintext into ciphertext.	Identify basic methods for transforming plaintext into ciphertext.	Describe basic methods for transforming plaintext into ciphertext, such as bit stream and block cipher.	Implement basic methods for transforming plaintext into ciphertext, such as bit stream and block cipher.
4. Explain the difference between symmetric and asymmetric encryption and how they are collectively used to secure digital	Recognize the difference between symmetric and asymmetric encryption	Explain the difference between symmetric and asymmetric encryption and how	Contrast symmetric and asymmetric encryption and their use in secure digital

communications and e-commerce transactions.	and how they are collectively used to secure digital communications and e-commerce transactions.	they are collectively used to secure digital communications and e-commerce transactions.	communications and e-commerce transactions.
IAS/Web Security Knowledge Unit			
1. Explain browser and web security model concepts including same-origin policy, web sessions, and secure communication channels.	Identify one or more browser and web security model concepts.	Explain browser and web security model concepts including same-origin policy, web sessions, and secure communication channels such as TLS.	Apply browser and web security model concepts including same-origin policy, web sessions, and secure communication channels such as TLS.
2. Investigate common vulnerabilities and attacks in web applications and the coding strategies that are used to mitigate them.	Describe common vulnerabilities and attacks in web applications and the coding strategies that are used to mitigate them.	Investigate common vulnerabilities and attacks in web applications and the coding strategies that are used to mitigate them.	Examine common vulnerabilities and attacks in web applications and the coding strategies that are used to mitigate them.
IAS/Secure Software Engineering Knowledge Unit			
1. Write software requirements that include basic security specifications.	Interpret software requirements that include basic security specifications.	Write software requirements that include basic security specifications.	Evaluate software requirements that include basic security specifications.
2. Implement a plan to test the security modules in software.	Summarize a plan to test the security modules in software.	Implement a plan to test the security modules in software.	Integrate a plan to test the security modules in software into the overall test plan.
3. Investigate security vulnerabilities in a software at the requirement and design phases of the software development life cycle.	Discuss security vulnerabilities in a software at the requirement and design phases of the software development life cycle.	Investigate security vulnerabilities in a software at the requirement and design phases of the software development life cycle.	Analyze security vulnerabilities in a software at the requirement and design phases of the software development life cycle.

Learning Outcome	Assessment Rubric		
IM/Information Management KA	**Emerging**	**Developed**	**Highly Developed**
IM/Information Management Concepts Knowledge Unit			
1. Describe how humans gain access to information and data to support their needs.	List ways in which humans gain access to information and data to support their needs.	Describe how humans gain access to information and data to support their needs.	Illustrate ways in which humans gain access to information and data to support their needs.
2. Describe the advantages and disadvantages of central organizational control over data	List the advantages and disadvantages of central organizational control over data.	Describe the advantages and disadvantages of central organizational control over data.	Assess the advantages and disadvantages of central organizational control over data.
3. Summarize the careers/roles associated with information management.	Identify the careers/roles associated with information management.	Summarize the careers/roles associated with information management.	Categorize the careers/roles associated with information management.
4. Differentiate information with data.	Define information and data.	Differentiate information and data.	Compare information with data.
5. Describe potential system attacks and the actors that might perform them.	Identify potential system attacks and the actors that might perform them.	Describe potential system attacks and the actors that might perform them.	Categorize potential system attacks and the actors that might perform them.
6. Describe contingency plans for various size organizations to include: business continuity, disaster recovery and incident response.	Recognize contingency plans for various size organizations to include: business continuity, disaster recovery and incident response.	Describe contingency plans for various size organizations to include: business continuity, disaster recovery and incident response.	Compare contingency plans for various size organizations to include: business continuity, disaster recovery and incident response.
7. Describe specific plans to secure data and information.	Recognize specific plans to secure data and information.	Describe specific plans to secure data and information.	Modify specific plans to secure data and information.
8. Discuss vulnerabilities and failure scenarios in common forms of information systems.	Identify vulnerabilities and failure scenarios in common forms of information systems.	Discuss vulnerabilities and failure scenarios in common forms of information systems e.g. SQL injection, cross-site scripting, etc.	Analyze vulnerabilities and failure scenarios in common forms of information systems.

IM/Database Systems Knowledge Unit			
1. Explain the characteristics that distinguish the database approach from the approach of programming with data files.	Identify the characteristics that distinguish the database approach from the approach of programming with data files.	Explain the characteristics that distinguish the database approach from the approach of programming with data files.	Contrast the characteristics that distinguish the database approach from the approach of programming with data files.
2. Describe the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor.	Define the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor.	Describe the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor.	Categorize the most common designs for core database system components including the query optimizer, query executor, storage manager, access methods, and transaction processor.
3. Summarize the basic goals, functions, and models of database systems.	Identify the basic goals, functions, and models of database systems.	Summarize the basic goals, functions, and models of database systems.	Assess the basic goals, functions, and models of database systems.
4. Describe the components of a database system and give examples of their use.	Identify the components of a database system.	Describe the components of a database system and give examples of their use.	Diagram the components of a database system and give examples of their use.
5. Describe the roles of major DBMS functions in a database system.	Identify major DBMS functions in a database system.	Describe the roles of major DBMS functions in a database system.	Assess major DBMS functions and their roles in a database system.
6. Explain the concept of data independence and its importance in a database system.	Define the concept of data independence and its importance in a database system.	Explain the concept of data independence and its importance in a database system.	Critique the concept of data independence and its importance in a database system.
7. Use a declarative query language to elicit information from a database.	Describe using a declarative query language to elicit information from a database.	Use a declarative query language to elicit information from a database.	Evaluate declarative query language used to elicit information from a database.
8. Describe common security concerns in database management systems.	Identify common security concerns in database management systems.	Describe common security concerns in database management systems.	Assess common security concerns in database management systems.
9. Apply security principles to the design and development of	Discuss security principles in the design	Apply security principles to the design	Assess security principles used in the

database systems and database structures.	and development of database systems and database structures.	and development of database systems and database structures.	design and development of database systems and database structures.
IM/Data Modeling Knowledge Unit			
1. Contrast appropriate data models, including internal structures, for different types of data.	Diagram appropriate data models, including internal structures, for different types of data.	Contrast appropriate data models, including internal structures, for different types of data.	Evaluate appropriate data models, including internal structures, for different types of data.
2. Describe concepts in modeling notation (e.g., Entity-Relation Diagrams or UML) and how they would be used.	Define concepts in modeling notation (e.g., Entity-Relation Diagrams or UML) and how they would be used.	Describe concepts in modeling notation (e.g., Entity-Relation Diagrams or UML) and how they would be used.	Illustrate concepts in modeling notation (e.g., Entity-Relation Diagrams or UML) and how they would be used.
3. Explain the fundamental terminology used in the relational data model.	Define the fundamental terminology used in the relational data model.	Explain the fundamental terminology used in the relational data model.	Use fundamental terminology in a relational data model.
4. Describe the basic principles of the relational data model.	Define the basic principles of the relational data model.	Describe the basic principles of the relational data model.	Apply basic principles in a relational data model.
5. Apply the modeling concepts and notation of the relational data model.	Explain the modeling concepts and notation of the relational data model.	Apply the modeling concepts and notation of the relational data model.	Examine the modeling concepts and notation of the relational data model.
6. Diagram a relational data model for a given scenario.	Describe a relational data model for a given scenario.	Diagram a relational data model for a given scenario.	Analyze a relational data model for a given scenario.
7. Describe the basic concepts of the OO model.	Identify the basic concepts of the OO model.	Describe the basic concepts of the OO model.	Apply the basic concepts of the OO model.
8. Describe the differences between relational data models and other models such as semi-structured or flexible schema (e.g., JSON, NoSQL).	Recognize the differences between relational data models and other models such as semi-structured or flexible schema (e.g., JSON, NoSQL).	Describe the differences between relational data models and other models such as semi-structured or flexible schema (e.g., JSON, NoSQL).	Investigate the differences between relational data models and other models such as semi-structured or flexible schema (e.g., JSON, NoSQL).
9. Describe relevant security and privacy issues given a system and data management structure.	Identify relevant security issues in a given system and data management structure.	Describe relevant security issues in a given system and data management structure.	Explore and illustrate relevant security issues in a given system and data management structure.

Learning Outcome	Assessment Rubric		
NC. Networking and Communication KA	**Emerging**	**Developed**	**Highly Developed**
NC/Introduction Knowledge Unit			
1. Explain the basic structure of the Internet.	Identify the basic structure of the Internet.	Explain the basic structure of the Internet.	Diagram the basic structure of the internet.
2. Define basic network terminology.	Recognize basic network terminology.	Define basic network terminology, such as topologies and protocols.	Describe basic network terminology.
3. Describe the layered structure of a typical networked architecture.	Label the components of a typical networked architecture.	Describe the layered structure of a typical networked architecture.	Diagram the layered structure of a typical networked architecture.
4. Diagram the layers of the OSI model.	Describe the layers of the OSI model.	Diagram the layers of the OSI model.	Compare and contrast the layers of the OSI model with the TCP/IP model.
NC/Networked Applications Knowledge Unit			
1. List the differences and the relationships between names and addresses in a network.	Identify one or more differences and relationships between names and addresses in a network.	List the differences and the relationships between names and addresses in a network.	Discuss the differences and the relationships between names and addresses in a network.
2. Define the principles behind naming schemes and resource location.	Define at least one principle behind naming schemes and resource location.	Define the principles behind naming schemes and resource location.	Demonstrate the principles behind naming schemes and resource location.
3. Implement a simple client-server socket-based application.	Identify the components of a simple client-server socket-based application.	Implement a simple client-server socket-based application.	Integrate a simple client-server socket-based application within a server program to exchange data with a client.

NC/Routing and Forwarding Knowledge Unit			
1. Describe how packets are forwarded in an IP network.	Recognize how packets are forwarded in an IP network.	Describe how packets are forwarded in an IP network.	Illustrate how packets are forwarded in an IP network.
2. Differentiate between routing and switching.	Recognize that routing and switching are not the same.	Differentiate between routing and switching.	Compare and contrast routing and switching.
NC/Local Area Networks Knowledge Unit			
1. Describe how frames are forwarded in a Local Area Network.	Recognize how frames are forwarded in a Local Area Network.	Describe how frames are forwarded in a Local Area Network.	Illustrate how frames are forwarded in a Local Area Network.
2. Describe the resources and services that Local Area Networks support.	List resources and services that Local Area Networks support.	Describe the resources and services Local Area Networks support, such as DHCP and DNS.	Examine resources and services in Local Area Networks.
NC/Mobility Knowledge Unit			
1. Describe the organization of a wireless network.	Identify components of a wireless network.	Describe the organization of a wireless network.	Diagram the organization of a wireless network.
2. Describe how wireless networks support mobile users.	Recognize how wireless networks support mobile users.	Describe how wireless networks support mobile users.	Implement wireless network support for mobile users.

Learning Outcome	**Assessment Rubric**		
OS. Operating Systems KA	**Emerging**	**Developed**	**Highly Developed**
OS/Overview of Operating Systems Knowledge Unit			
1. Explain major objectives, functions and concepts of modern operating systems.	List some objectives, functions and concepts of modern operating systems.	Explain major objectives, functions, and concepts of modern operating systems.	Explain in detail most objectives, functions and concepts of modern operating systems.

2. Describe key features of a contemporary operating system, such as scripting, user interfaces, and updates, with respect to convenience, efficiency, and the ability to evolve.	Identify key features of a contemporary operating system, such as scripting, user interfaces, and updates, with respect to convenience, efficiency, and the ability to evolve.	Describe key features of a contemporary operating system, such as scripting, user interfaces, and updates, with respect to convenience, efficiency, and the ability to evolve.	Use key features of a contemporary operating system, such as scripting, user interfaces, and updates, with respect to convenience, efficiency, and the ability to evolve.
3. Differentiate between prevailing types of operating systems, such as networked, mobile, real-time, and distributed.	Define prevailing types of operating systems, such as networked, mobile, real-time, and distributed.	Differentiate between prevailing types of operating systems, such as networked, mobile, real-time, and distributed.	Investigate prevailing types of operating systems, such as networked, mobile, real-time, and distributed.
4. Discuss potential threats to operating systems and the security features designed to guard against them.	Identify some potential threats to operating systems.	Discuss potential threats to operating systems and the security features designed to guard against them.	Investigate potential threats to operating systems and the security features designed to guard against them.
OS/Operating System Principles Knowledge Unit			
1. Describe the value of APIs and middleware.	Identify the basic functions of APIs and middleware.	Describe the value of APIs and middleware.	Use APIs and middleware.
2. Describe how computing resources are used by application software and managed by system software.	Identify which computing resources are used by application software and managed by system software.	Describe how computing resources are used by application software and managed by system software.	Investigate how computing resources are used by application software and managed by system software.
3. Define a device list and a driver I/O queue.	Recognize a device list and driver I/O queue.	Define a device list and a driver I/O queue.	Explain the use of a device list and driver I/O queue.
OS/Concurrency Knowledge Unit			
1. Describe the need for concurrency within the framework of an operating system.	Define concurrency within the framework of an operating system.	Describe the need for concurrency within the framework of an operating system.	Investigate the need for concurrency within the framework of an operating system.
OS/Memory Management Knowledge Unit			
1. Describe the principles of memory management, including	Define the principles of memory	Describe the principles of memory	Illustrate the principles of memory

the function of and need for cache memory.	management, including the function of and need for cache memory.	management, including the function of and need for cache memory.	management, including the function of and need for cache memory.
2. Describe the principles of virtual memory as applied to caching and paging.	List some of the principles of virtual memory as applied to caching and paging.	Describe the principles of virtual memory as applied to caching and paging.	Explain the principles of virtual memory as applied to caching and paging.
3. Define the concept of thrashing.	Recognize the concept of thrashing.	Define the concept of thrashing.	Explain the concept of thrashing and how to manage it.
OS/Security and Protection Knowledge Unit			
1. Explain the need for protection and security in an OS.	Identify the need for protection and security in an OS.	Explain the need for protection and security in an OS.	Investigate the need for protection and security in an OS.
2, Summarize the features of an operating system used to provide protection and security.	Describe some of the features of an operating system used to provide protection and security.	Summarize the features and limitations of an operating system used to provide protection and security.	Implement some of the features of an operating system used to provide protection and security.
3. Explain the mechanisms available in an OS to control access to resources.	List some of the mechanisms available in an OS to control access to resources.	Explain the mechanisms available in an OS to control access to resources.	Implement some of the mechanisms available in an OS to control access to resources.
OS/Virtual Machines Knowledge Unit			
1. Explain the concept of virtualization and how it is realized in hardware and software.	Define the concept of virtualization and how it is realized in hardware and software.	Explain the concept of virtualization and how it is realized in hardware and software.	Investigate the concept of virtualization and how it is realized in hardware and software.
OS/Device Management Knowledge Unit			
1. Explain the relationship between the physical hardware and the virtual devices maintained by the operating system.	Identify the difference between physical hardware and virtual devices.	Explain the relationship between the physical hardware and the virtual devices maintained by the operating system.	Diagram the relationship between the physical hardware and the virtual devices maintained by the operating system.

Learning Outcome	Assessment Rubric		
PL. Programming Languages KA	**Emerging**	**Developed**	**Highly Developed**
PL/Object-Oriented Programming Knowledge Unit			
1. Implement a simple class hierarchy, including superclasses and subclasses, using encapsulation, abstraction, inheritance, and polymorphism.	Describe a simple class hierarchy, including superclasses and subclasses, using encapsulation, abstraction, inheritance, and polymorphism.	Implement a simple class hierarchy, including superclasses and subclasses, using encapsulation, abstraction, inheritance, and polymorphism.	Analyze a simple class hierarchy, including superclasses and subclasses, using encapsulation, abstraction, inheritance, and polymorphism.
2. Use control flow in a program using dynamic dispatch.	Describe the best ways to use flow control in a program using dynamic dispatch.	Use control flow in a program using dynamic dispatch.	Contrast control flow using a static environment vs a dynamic environment.
3. Develop secure GUI applications using modern GUI development libraries and tools.	Explain how GUI applications can be developed using modern GUI development libraries and tools.	Develop secure GUI applications using modern GUI development libraries and tools.	Evaluate the outcomes of a GUI based on a give input to interact with the reactive system.
4. Use private and protected methods to secure class data and to demonstrate encapsulation.	Describe private and protected methods and their role in encapsulation and the security of class data.	Use private and protected methods to secure class data and to demonstrate encapsulation.	Analyze private and protected methods in relation to encapsulation and security of class data.
5. Apply fundamental security principles and strategies to software development to inhibit attacks.	Describe different software development security principles.	Use proper software development security principles to write a small program.	Design a program using good secure software development techniques.
6. Illustrate the secure development lifecycle.	Explain the secure development lifecycle.	Illustrate the secure development lifecycle.	Analyze the secure development lifecycle.
7. Assess secure coding by checking parameters based on certain restrictions on data.	Understand the concept of secure coding.	Assess secure coding by checking parameters based on certain restrictions on data.	Develop a program using the secure coding methodology.
8. Describe the tenets of OOP: encapsulation, abstraction,	List the tenets of OOP:such as	Describe the tenets of OOP: encapsulation,	Investigate existing code and the impact

inheritance, and polymorphism and how they impact security.	encapsulation, abstraction, inheritance, and polymorphism and how they are used to impact security.	abstraction, inheritance, and polymorphism and how they impact security.	encapsulation, abstraction, inheritance, and polymorphism have on security.
9. Describe the characteristics of static, stack, and heap allocation.	Identify local variables within the scope of a method as well as a global variable such as a class or main class variable.	Describe the characteristics of static, stack, and heap allocation.	Implement a program in such a way the main method tries to access global, local, and instance variables.
10. Describe the state of the call stack when calling non-recursive and recursive subroutines.	Recognize that recursion is an optional feature in OOP languages such as Java, however is not part of the paradigm principles.	Describe the state of the call stack when calling non-recursive and recursive subroutines.	Illustrate the activation records storage by tracing recursive calls until a method reaches its base-case.
11. Explain the difference between method definition and method calling.	Translate a instructions from a text and translate it to a method definition based on the description of the problem.	Explain the difference between method definition and method calling.	Test methods with different values as a parameters to distinguish between method calls and method definitions.
12. Describe confidentiality, integrity, and availability and the impact each has on security.	Define confidentiality, integrity, and availability.	Describe confidentiality, integrity, and availability and the impact each has on security.	Illustrate confidentiality, integrity, and availability and the impact each has on security.
PL/Functional Programming Knowledge Unit			
1. Write basic algorithms that avoid assigning to mutable state or considering reference equality.	Discuss how in functional languages, rather than mutating the state of objects, the objects are simply returned with the desired reflected changes.	Write basic algorithms that avoid assigning to mutable state or considering reference equality.	Evaluate efficiency by comparing algorithms that avoid assigning to mutable state vs modifying data structures values.
2.Compare and contrast the procedural/functional approach and the object-oriented approach.	Differentiate between different programming paradigms by mention advantages and disadvantages of each	Compare and contrast the procedural/functional approach and the object-oriented	Apply a function for a given operation with the function body providing a case for each data variant.

	paradigm.	approach.	
3. Write useful functions that take and return other functions.	Explain what Lambda calculus is and how this principle is used in computer programming.	Write useful functions that take and return other functions.	Test several functions with different definitions to produce a desired output based on returning other functions.
PL/Event-Driven and Reactive Programming Knowledge Unit			
1. Write event handlers for use in reactive systems, such as GUIs.	Design a test case scenario in how to interact with a reactive system; discuss potential states that the system may have.	Write event handlers for use in reactive systems, such as GUIs.	Evaluate outcomes from a GUI by providing a set of instructions and anticipate usability of the system.
2. Explain why an event-driven programming style is natural in domains where programs react to external events.	Describe an advantage of having an event-driven programming style vs a pre-defined programming style.	Explain why an event-driven programming style is natural in domains where programs react to external events.	Use an event-driven style to generate outcomes that can be evaluate them.
3. Describe an interactive system in terms of a model, a view, and a controller.	Define the notion of MVC and mention the advantages of this model.	Describe an interactive system in terms of a model, a view, and a controller.	Integrate several active components and design a reactive system that implements the MVC.
4. Describe how event-driven GUI applications are structured when guarding against injection-based attacks.	Demonstrate what injection-based attacks can happen under a piece of code that runs in response of an action.	Describe how event-driven GUI applications are structured when guarding against injection-based attacks.	Implement a GUI where multiple selections can be enabled in such a way that create a logic inconsistency on multiple values selected.
PL/Basic Type Systems Knowledge Unit			
1. Describe examples of program errors detected by a type system.	List a set of examples that occurred in the past because of a lack of strong type system.	Describe examples of program errors detected by a type system.	Implement a program that demonstrates type-system error.
2. For multiple programming languages, identify program properties checked statically and program properties checked dynamically.	Identify the notion and need for static vs dynamic usage.	For multiple programming languages, identify program properties checked statically and program properties checked dynamically.	Compare outcomes from different programs that define variables in the stack and heap; Discuss the impact to have these separations in

			memory.
3. Write an example program that does not type-check in a particular language and yet would have no error if run.	Modify a program that process different data types and perform operations within the same data.	Write an example program that does not type-check in a particular language and yet would have no error if run.	Test a program that given different data type arguments as input, perform arithmetic operations without causing an error.
4. Use types and type-error messages to write and debug programs.	Describe sandbox testing to handle potential type-errors when a data type exception occurs.	Use types and type-error messages to write and debug programs.	Test different cases when different type-error messages may occur by providing multiple inputs to a pre-defined methods in a program.
5. Describe how object-oriented languages which are strong-type languages define a data type in compile time.	Identify different data types capacities and compatibility between data types.	Describe how object-oriented languages which are strong-type languages define a data type in compile time.	Apply type casting in different data types to recognize compatibility and possible lost of precision in other cases.
6. Explain why you might choose to develop a program in a type-safe language in contrast to an unsafe programming language.	Recall notable catastrophic events that were produced by a type-safe error processing.	Explain why you might choose to develop a program in a type-safe language in contrast to an unsafe programming language.	Implement a program that keeps computing data with possible lost of precision; demonstrate the need of type-safe languages in real situations.

Learning Outcome	**Assessment Rubric**		
SDF. Software Development Fundamentals KA	**Emerging**	**Developed**	**Highly Developed**
SDF/Algorithms and Design Knowledge Unit			
1. Discuss the importance of algorithms in the problem-solving process.	List advantages of using algorithms in the problem-solving process.	Discuss the importance of algorithms in the problem-solving process.	Illustrate the importance of algorithms in the problem-solving process.

2. Discuss how a problem may be solved by multiple algorithms, each with different properties.	Identify a single algorithm to solve a problem.	Discuss how a problem may be solved by multiple algorithms, each with different properties.	Use multiple algorithms to help solve a problem.
3. Use a programming language to implement algorithms designed to solve simple problems.	Use a programming language to explain how to solve simple problems using an algorithm.	Use a programming language to implement algorithms designed to solve simple problems.	Use a programming language to test algorithms for solving simple problems.
4. Implement, test, and debug simple recursive functions and procedures.	Identify different types of recursive functions and procedures.	Implement, test, and debug simple recursive functions and procedures.	Write a program that uses different simple recursive functions and procedures to solve a problem.
5. Apply the techniques of decomposition to break a program into smaller pieces.	Define decomposition with regard to its use in computer science.	Use the techniques of decomposition to break a program up into smaller pieces.	Evaluate code to see how decomposition techniques were used.
6. Describe the data components and behaviors of multiple abstract data types.	List the data components and behaviors of multiple abstract data types.	Describe the data components and behaviors of multiple abstract data types.	Choose the proper abstract data type to be used in a program
7. Write simple programs which use abstract data types (ADTs).	Describe simple programs which use abstract data types (ADTs).	Write simple programs which use abstract data types (ADTs)..	Analyze simple programs which use abstract data types (ADTs).
8. Identify the relative strengths and weaknesses among multiple designs or implementations for a problem.	Given several samples of algorithms, state the strengths and weaknesses amongst them.	Illustrate different ways to write an algorithm for a specific task.	Develop a solid algorithm to solve a business problem.
9. Demonstrate brute-force algorithms vs divide and conquer algorithms to accomplish security objectives: e.g., attempt to breaking a password.	Explain the differences between brute-force algorithms versus divide and conquer algorithms and how they can help accomplish security objectives.	Demonstrate brute-force algorithms vs divide and conquer algorithms to accomplish security objectives: e.g., attempt to breaking a password.	Implement a multiple nested loop to create possible combinations of strings to match for a given password.
10. Explain the difference and complexity between iterative-based and recursive-based implementations.	Identify both iterative-based and recursive-based functions.	Explain the difference and complexity between iterative-based and recursive-based implementations when used to compute the	Evaluate the best use of iterative-based and recursive-based functions for a given problem.

		same task.	
11. Choose whether a recursive or iterative solution is most appropriate for a problem.	List several attributes that recursive and iterative-based algorithm have as approaches to solve a problem	Choose whether a recursive or iterative solution is most appropriate for a problem.	Evaluate the performance for two different implementations of the same problem
SDF/Fundamental Programming Concepts Knowledge Unit			
1. Describe the characteristics of secure programming.	List the characteristics of secure programming.	Differentiate between writing code using the characteristics of secure coding versus just coding.	Write a program using secure programming methods to solve a business problem.
2. Discuss the importance of usability in security mechanism design.	Define what is security mechanism design.	Discuss the importance of usability in security mechanism design.	Analyze programs to see of there are any security mechanism design failures, and correct them.
3. Summarize the principle of fail-safe and deny-by-default.	Define the principles of fail-safe and deny-by-default.	Summarize the principle of fail-safe and deny-by-default.	Use the principles of fail-safe and deny-by-default in an algorithm to solve a problem.
4. Produce software components that satisfy their functional requirements without introducing vulnerabilities.	Explain what potential vulnerabilities may exist when creating software.	Produce software components that satisfy their functional requirements without introducing vulnerabilities.	Evaluate code to see if there are any vulnerabilities introduced in the way the code is written and executed.
5. Write code which uses defensive programming methods, such as input validation, type checking and buffer overflow.	Describe code which uses defensive programming methods, such as input validation, type checking and buffer overflow.	Write code which uses defensive programming methods, such as input validation, type checking and buffer overflow.	Examine code which uses defensive programming methods, such as input validation, type checking and buffer overflow.
6. Examine security objectives by identifying bad input from the user according to the program design.	List issues with security by identifying bad input from the user.	Examine security objectives by identifying bad input from the user according to the program design.	Assess security objectives by identifying bad input from the user according to the program design.
SDF/Fundamental Data Structures Knowledge Unit			
1. Describe common applications for each of the	List common applications for each	Describe common applications for each	Investigate common applications for each

following data structures: stack, queue, priority queue, set, and map.	of the following data structures: stack, queue, priority queue, set, and map.	of the following data structures: stack, queue, priority queue, set, and map.	of the following data structures: stack, queue, priority queue, set, and map.
2. Write programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps.	Define the following data structures: arrays, records/structs, strings, linked lists,stacks, queues, sets, and maps.	Write programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps.	Analyze programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps.
3. Compare alternative implementations of data structures with respect to performance.	Describe alternative implementations of data structures with respect to performance.	Compare alternative implementations of data structures with respect to performance.	Evaluate alternative implementations of data structures with respect to performance.
4. Describe how references allow for objects to be accessed in multiple ways.	Define a class that contains several fields and methods that can be used to calculate certain information.	Describe how references allow for objects to be accessed in multiple ways.	Implement a tester that initializes several instances of a given object, store them into an array of the same type and print the array contents.
5. Choose the appropriate data structure to solve a problem.	Define built-in data structures.	Choose the appropriate data structure to solve a problem.	Write a program that uses at least three different data structures.
SDF/Development Methods Knowledge Unit			
1. Describe common coding errors that introduce security vulnerabilities.	List common coding errors that introduce security vulnerabilities.	Describe common coding errors that introduce security vulnerabilities. (e.g., buffer overflows, memory leaks, malicious code).	Describe common coding errors that introduce security vulnerabilities and the associated techniques for securing the code.
2. Perform a code review on a program component.	Describe a code review on a program component.	Perform a code review on a program component (e.g. secure, correct, complete).	Outline a code review on a program component.
3. Describe opportunities within given program components for simple refactoring.	List opportunities within given program components for simple refactoring.	Describe opportunities within given program components for simple refactoring.	Implement simple refactoring within given program components.

4. Describe contract programming and the role of preconditions, postconditions, and invariants.	List the importance concepts associated with contract programming.	Describe contract programming and the role of preconditions, postconditions, and invariants.	Illustrate contract programming and the role of preconditions, postconditions, and invariants for a programming component.
5. Apply a variety of strategies to test and debug simple programs.	List strategies to test and debug simple programs.	Apply a variety of strategies to test and debug simple programs (e.g., unit testing, test-case generation).	Analyze the effectiveness of a variety of strategies to test and debug simple programs.
6. Use an IDE to create, execute, and debug programs.	Discuss the benefits of using an IDE to create, execute, and debug programs.	Use an IDE to create, execute, and debug programs.	Compare IDEs which can be used to create, execute, and debug programs for a given programming language.
7. Use standard libraries for a given programming language to create, execute, and debug programs.	List the standard libraries for a given programming.	Use standard libraries for a given programming language to create, execute, and debug programs.	Examine the standard libraries for a given programming language.
8. Apply consistent documentation and program style standards.	Discuss the pros and cons of inconsistent documentation and program style standards.	Apply consistent documentation and program style standards.	Evaluate documentation and program style in a given coding solution.

Learning Outcome	**Assessment Rubric**		
SE. Software Engineering KA	**Emerging**	**Developed**	**Highly Developed**
SE/Software Processes Knowledge Unit			
1. Describe how software can interact with and participate in various systems including information management, embedded, process control, and communications systems.	Recognize how software can interact with and participate in various systems including information management, embedded, process	Describe how software can interact with and participate in various systems including information management, embedded, process	Implement software that can interact with and participate in various systems including information management, embedded, process

	control, and communications systems.	control, and communications systems.	control, and communications systems.
2. Describe the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile).	Identify the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile).	Describe the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile).	Contrast the relative advantages and disadvantages among several major process models (e.g., waterfall, iterative, and agile).
3. Describe the different practices that are key components of various process models.	List the different practices that are key components of various process models.	Describe the different practices that are key components of various process models.	Apply the different practices that are key components of various process models.
4. Differentiate among the phases of software development.	List the phases of software development.	Differentiate among the phases of software development.	Perform the phases of software development.
5. Describe how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes.	List how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes.	Describe how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes.	Illustrate how programming in the large differs from individual efforts with respect to understanding a large code base, code reading, understanding builds, and understanding context of changes.
6. Describe the relative advantages and disadvantages among several major process models.	Identify the relative advantages and disadvantages among several major process models.	Describe the relative advantages and disadvantages among several major process models (e.g., multi-level security, waterfall with security, comprehensive lightweight application security process (CLASP), Extreme Programming, Aspect-oriented programming).	Use one of the several major process models.
SE/Software Project Management Knowledge Unit			
1. Discuss common behaviors that contribute to the effective functioning of a team.	List common behaviors that contribute to the effective functioning of a team.	Discuss common behaviors that contribute to the effective functioning of a team.	Examine common behaviors that contribute to the effective functioning of a team.

2. Describe necessary roles in a software development team.	Identify necessary roles of a software development team.	Describe necessary roles in a software development team.	Compare necessary roles in a software development team.
3. Describe the sources, hazards, and potential benefits of team conflict.	List the sources, hazards, and potential benefits of team conflict.	Describe the sources, hazards, and potential benefits of team conflict.	Illustrate the sources, hazards, and potential benefits of team conflict.
4. Apply a conflict resolution strategy in a team setting.	Describe conflict resolution strategies for a team.	Apply a conflict resolution strategy in a team setting.	Contrast conflict resolution strategies for teams.
5. Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required	Describe an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required	Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required	Evaluate an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required
6. Describe different categories of risk in software systems.	List different categories of risk in software systems.	Describe different categories of risk in software systems.	Contrast different categories of risk in software systems.
7. Discuss the need to update software to fix security vulnerabilities and the life cycle management of the fix.	State the need to update software to fix security vulnerabilities and the life cycle management of the fix.	Discuss the need to update software to fix security vulnerabilities and the life cycle management of the fix.	Implement updates to software to fix security vulnerabilities
SE/Tools and Environments Knowledge Unit			
1. Describe the difference between centralized and distributed software configuration management.	Define centralized and distributed software configuration management.	Describe the difference between centralized and distributed software configuration management.	Contrast the difference between centralized and distributed software configuration management.
2. Describe how version control can be used to help with software release management	Identify version control technologies that can be used for software release management	Describe how version control can be used to help with software release management	Use version control for software release management
3. Explain configuration items and how to use a source code control tool in a small team-based project.	Identify configuration items and a source code control tool in a small team-based project.	Explain configuration items and how to use a source code control tool in a small team-based project.	Modify configuration items and use a source code control tool in a small team-based project.
4. Describe how available static and dynamic test tools can be integrated into the software development environment.	List available static and dynamic test tools that can be integrated into the software	Describe how available static and dynamic test tools can be integrated into the	Use static and dynamic test tools in a software development environment.

	development environment.	software development environment.	
5. Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing.	List the factors that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing.	Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing.	Investigate the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing.
6. Demonstrate the capability to use software tools in support of the development of a software product of medium size.	Identify software tools to support the development of a software product of medium size.	Demonstrate the capability to use software tools in support of the development of a software product of medium size.	Use software tools in support of the development of a software product of medium size.
7. Use a modern IDE and debugger for security-minded debugging and testing.	Explain how a modern IDE and debugger can be used for security-minded debugging and testing.	Use a modern IDE and debugger for security-minded debugging and testing.	Assess a modern IDE and debugger for security-minded debugging and testing.
8. Explain the risks with misusing interfaces with third-party code and how to correctly use third-party code.	List the risks with misusing interfaces with third-party code and how to correctly use third-party code.	Explain the risks with misusing interfaces with third-party code and how to correctly use third-party code.	Illustrate the risks with misusing interfaces with third-party code and how to correctly use third-party code.
9. Use static and dynamic tools to identify programming faults.	Identify static and dynamic tools to identify programming faults.	Use static and dynamic tools to identify programming faults.	Evaluate static and dynamic tools to identify programming faults.
SE/Requirements Engineering Knowledge Unit			
1. Describe the key components of a use case or similar description of some behavior that is required for a system.	List the key components of a use case or similar description of some behavior that is required for a system.	Describe the key components of a use case or similar description of some behavior that is required for a system.	Decompose use case into its key components.
2. Describe how the requirements engineering process supports the elicitation and validation of behavioral	Define the requirements engineering in the context of behavioral	Describe how the requirements engineering process supports the elicitation	Develop software requirements based on the elicitation and validation of

requirements.	requirements.	and validation of behavioral requirements.	behavioral requirements.
3. Interpret a given requirements model for a simple software system.	Recognize a given requirements model for a simple software system.	Interpret a given requirements model for a simple software system.	Produce a requirements model for a simple software system.
4. Write system requirements from a client concept/specification that incorporate threat models.	Discuss system requirements from a client concept/specification that incorporate threat models.	Write system requirements from a client concept/specification that incorporate threat models.	Analze system requirements from a client concept/specification that incorporate threat models.
5. Write user narratives in preparation for building a piece of software.	Classify user narratives in preparation for building a piece of software.	Write user narratives in preparation for building a piece of software.	Analyze use-cases from user narratives in preparation for building a piece of software.
6. Describe important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing.	List important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing.	Describe important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing.	Categorize important ethical issues to consider in computer security, including ethical issues associated with fixing or not fixing.
7. Describe the concepts of risk, threats, vulnerabilities and attack vectors (including the fact that there is no such thing as perfect security).	Identify the concepts of risk, threats, vulnerabilities and attack vectors (including the fact that there is no such thing as perfect security).	Describe the concepts of risk, threats, vulnerabilities and attack vectors (including the fact that there is no such thing as perfect security).	Assess the concepts of risk, threats, vulnerabilities and attack vectors of software requirements (including the fact that there is no such thing as perfect security).
8. Explain the concept of trust and trustworthiness.	Define the concept of trust and trustworthiness.	Demonstrate the concept of trust and trustworthiness.	Assess the concept of trust and trustworthiness in software requirements.
9. Explain the concepts of authentication, authorization, access control.	Define the concepts of authentication, authorization, access control.	Explain the concepts of authentication, authorization, access control.	Use the concepts of authentication, authorization, access control in software requirements.

SE/Software Design Knowledge Unit			
1. Describe different system design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures.	Recognize different system design principles.	Describe different system design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures.	Illustrate different system design principles: levels of abstraction (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures.
2. Analyze an existing software implementation and make suggestions to improve security in its design.	Identify possible stages of software design that may introduce potential information leakage or a security vulnerability.	Analyze an existing software implementation and make suggestions to improve security in its design.	Debate whether a proposed solution/patch to the design can fix the vulnerability in a viable and effective way.
3. Implement security improvements in an existing software implementation.	Locate the code from existing software that generates the security vulnerability .	Implement security improvements in an existing software implementation.	Justify an improvement of the security vulnerability in an existing software implementation.
4. Describe standard components for security operations, and explain the benefits of their use instead of re- inventing fundamentals operations.	Identify standard components for security operations.	Describe standard components for security operations, and explain the benefits of their use instead of re-inventing fundamentals operations.	Categorize standard components for security operations based on benefits of their use.
5. Describe the concept of mediation and the principle of complete mediation.	Identify cases where the concept of mediation and the principle of complete mediation can be used in the software design process.	Describe the concept of mediation and the principle of complete mediation.	Evaluate the usability design two models based on the concept of mediation and the principle of complete mediation.
6. Describe the cost and tradeoffs associated with designing security into a product.	Recognize situations where security designs are effectively applied in a product.	Describe the cost and tradeoffs associated with designing security into a product.	Compare security designs and their current costs and tradeoffs.
7. Describe the requirements for integrating security into the software development	Identify potential stages of the software development lifecycle	Describe the requirements for integrating security	Analyze the impact in a software life cycle of integrating a security

lifecycle.	where security is needed.	into the software development lifecycle.	component.
8. Explain the concept of trusted computing including trusted computing base and attack surface and the principle of minimizing trusted.	Define concepts of trusted computing, attack surface, and minimizing trusted computing .	Explain the concept of trusted computing including trusted computing base and attack surface and the principle of minimizing trusted .	Illustrate cases where these concepts can be applied in the design process .
SE/Software Construction Knowledge Area			
1. Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness.	Identify techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness.	Describe techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness.	Implement techniques, coding idioms and mechanisms for implementing designs to achieve desired properties such as reliability, efficiency, and robustness.
2. Classify a robust code using exception handling mechanisms.	Recognize potential input that may cause an exception in the code and handle the output effectively using a exception handling mechanism.	Classify a robust code using exception handling mechanisms.	Implement a program that handles a potential exception that may be caused by an invalid input.
3. Describe secure coding and defensive coding practices.	List several strategies for defensive coding.	Describe secure coding and defensive coding practices.	Implement a defensive coding practice.
4. Analyze a retired system for actionable attack information.	Discuss a retired system for potential attack vulnerabilities.	Analyze a retired system for actionable attack information.	Compare different attack information from a retired system.
5. Describe the process of analyzing and implementing changes to code base developed for a specific project.	Discuss the importance of software construction based on changing pieces of code to be compatible with other components of a specific project.	Describe the process of analyzing and implementing changes to code base developed for a specific project.	Implement specific changes to an existing code to operate with other components of a specific project.
6. Edit a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions, and test to insure the components are resilient to input and run-time	Identify a list of common vulnerabilities in a simple program.	Edit a simple program to remove common vulnerabilities, such as buffer overflows, integer overflows and race conditions, and test to insure the	Evaluate a simple program in which vulnerabilities were removed .

errors.		components are resilient to input and run-time errors.	
7. Diagram use cases.	Describe the use case for each sta.ge on the lifecycle	Diagram use cases.	Examine use cases for a given task.
8. Examine the vulnerabilities in a given development process to an insider inserting undetected backdoor insertion.	Describe the potential vulnerabilities that may occur in a given development process .	Examine the vulnerabilities in a given development process to an insider inserting undetected backdoor insertion.	Verify that a current solution is effective in a given development process.
9. Use a defined coding standard in a small software project.	Select a defined coding standard in a small software project.	Use a defined coding standard in a small software project.	Justify the reason of using a given code.
SE/Software Verification and Validation Knowledge Area			
1. Distinguish between program validation and verification.	Explain the difference between validation and verification of data	Distinguish between program validation and verification.	Implement a piece of code that validates and verifies a given input
2. Describe the role that tools can play in the validation of software.	Recognize well-known tools that validate certain software.	Describe the role that tools can play in the validation of software.	Use tools that help validate the input of a software.
3. Describe among the different types and levels of testing (unit, integration, systems, and acceptance).	List the different types and levels of testing	Describe among the different types and levels of testing (unit, integration, systems, and acceptance).	Distinguish among the different types and levels of testing (unit, integration, systems, and acceptance).
4.Describe techniques to identify and select optimal and significant test cases for integration, regression and system testing.	Define integration, regression and system testing.	Describe techniques to identify and select optimal and significant test cases for integration, regression and system testing.	Compare several techniques for identifying significant test cases for integration, regression and system testing.
5. Use a defect tracking tool to manage software defects in a small software project.	List tracking tools used to manage software defects	Use a defect tracking tool to manage software defects in a small software project.	Evaluate tracking tools that manage software defects
6. Discuss the limitations of testing in a particular domain.	Explain the importance of not restrict the testing of a code to a specific domain	Discuss the limitations of testing in a particular domain.	Categorize the limitations of testing in a particular domain.

7. Describe input validation and data sanitization including how code is tested for input handling errors and the impact this has on security.	Define input validation and data sanitization.	Describe input validation and data sanitization including how code is tested for input handling errors and the impact this has on security.	Write code that performs input validation and data sanitization.
SE/Software Evolution Knowledge Unit			
1. Identify the principal issues associated with software evolution and explain their impact on the software lifecycle.	Recognize the stages of the software lifecycle where software evolution can take place.	Identify the principal issues associated with software evolution and explain their impact on the software lifecycle.	Implement software evolution in given stages of the software lifecycle .
2. Use refactoring in the process of modifying a software component.	Explain the existence of design patterns in software development.	Use refactoring in the process of modifying a software component.	Implement a refactoring pattern to modify a software component.
3. Discuss the challenges of evolving systems in a changing environment.	Define what evolving systems are.	Discuss the challenges of evolving systems in a changing environment.	Analyze existing systems in known controlled environments.
4. Outline the process of regression testing and its role in release management.	Identify the stage of software lifecycle where regression testing takes place.	Outline the process of regression testing and its role in release management.	Assess output obtained from a process of regression testing.
5. Discuss the advantages and disadvantages of different types of software reuse.	Define software reusability.	Discuss the advantages and disadvantages of different types of software reuse.	Integrate components from previous pieces of code and contrast the outcomes from current version.
6. Describe software development best practices for minimizing vulnerabilities in programming code.	List several vulnerabilities that may occur in the process of software development.	Describe software development best practices for minimizing vulnerabilities in programming code.	Evaluate best practices for minimizing vulnerabilities in programming code.
SE/Software Reliability Knowledge Unit			
1. Explain the problems that exist in achieving very high levels of reliability.	Identify the trade-off of achieving very high levels of reliability.	Explain the problems that exist in achieving very high levels of reliability.	Analyze outcomes of a software by obtaining a high level of reliability.
2. Describe how software reliability contributes to system reliability.	Recognize trade-offs between software reliability and system	Describe how software reliability contributes to system	Justify how software reliability contributes to system reliability.

	reliability.	reliability.	
3. Identify ways to apply redundancy to achieve fault tolerance for a medium-sized application.	Recognize locations to apply redundancy for a medium-sized application.	Identify ways to apply redundancy to achieve fault tolerance for a medium-sized application.	Analyze the fault tolerance for an application.
4. Analyze OOP design patterns and explain how they impact reliability and security.	Recognize design patterns in the software lifecycle components .	Analyze OO design patterns and explain how they impact reliability and security.	Compare design patterns and their impact to achieve reliability and security in the software development.
5. List approaches to minimizing faults that can be applied at each stage of the software lifecycle.	Identify faults that can occur in different stages of the software lifecycle.	List approaches to minimizing faults that can be applied at each stage of the software lifecycle.	Evaluate the approaches used to minimizing faults at each stage of the software lifecycle.

Learning Outcome	**Assessment Rubric**		
SF. System Fundamentals KA	**Emerging**	**Developed**	**Highly Developed**
SF/Computational Paradigms Knowledge Unit			
1. List commonly encountered patterns of how computations are organized.	Recognize some patterns of how computations are organized.	List commonly encountered patterns of how computations are organized.	Explain commonly encountered patterns of how computations are organized.
2. Identify the basic building blocks of computers and their role in the historical development of computer architecture.	Identify some of the basic building blocks of computers.	Identify the basic building blocks of computers and their role in the historical development of computer architecture.	Discuss the basic building blocks of computers and their role in the historical development of computer architecture.
3. Discuss the differences between single thread and multiple thread, single server and multiple server models.	Identify some differences between single thread and multiple thread, single server and multiple server models.	Discuss the differences between single thread and multiple thread, single server and multiple server models.	Illustrate the differences between single thread and multiple thread, single server and multiple server models.
4. Illustrate performance of simple sequential and parallel versions of a program with	Discuss the general performance of simple sequential and parallel	Illustrate performance of simple sequential and parallel versions	Compare performance of sequential and parallel versions of a

different problem sizes.	versions of a program.	of a program with different problem sizes.	program with different problem sizes.
5. Recognize security implications related to emerging computational paradigms.	List one or more security implications related to computational paradigms.	Recognize security implications related to emerging computational paradigms, such as Quantum Computing.	Discuss security implications related to emerging computational paradigms.
SF/Cross-Layer Communications Knowledge Unit			
1. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers.	Recognize that computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers.	Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers.	Diagram a computing systems constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers.
2. Implement a simple program using methods of layering, error detection and recovery, and reflection of error status across layers.	Explain a simple program that uses methods of layering, error detection and recovery, and reflection of error status across layers.	Implement a simple program using methods of layering, error detection and recovery, and reflection of error status across layers.	Implement a complex program using methods of layering, error detection and recovery, and reflection of error status across layers.
3. Investigate bugs in a layered program using tools for program tracing, single stepping, and debugging.	Demonstrate bugs in a layered program using tools for program tracing, single stepping, and debugging.	Investigate bugs in a layered program using tools for program tracing, single stepping, and debugging.	Categorize bugs by security risk in a layered program using tools for program tracing, single stepping, and debugging.

Learning Outcome	Assessment Rubric		
SP. Social Issues and Professional Practice KA	**Emerging**	**Developed**	**Highly Developed**
SP/Social Context Knowledge Unit			
1. Describe positive and negative ways in which computer technology (networks, mobile computing, cloud computing) alters modes of social interaction at the personal level.	Identify ways in which computer technology (networks, mobile computing, cloud computing) affects social interaction at the personal level.	Describe positive and negative ways in which computer technology (networks, mobile computing, cloud computing) alters modes of social interaction at the personal level.	Critique positive and negative ways in which computer technology (networks, mobile computing, cloud computing) changes methods of social interaction at the personal level.
2. Interpret developers' assumptions and values embedded in hardware and software design, especially as they pertain to usability for diverse populations including under-represented populations and the disabled.	Identify developers' assumptions and values embedded in hardware and software design, especially as they pertain to underrepresented populations and the disabled.	Interpret developers' assumptions and values embedded in hardware and software design, especially as they pertain to usability for diverse populations including under-represented populations and the disabled.	Contrast developers' assumptions and values embedded in hardware and software design, especially as they pertain to usability for diverse populations including under-represented populations and the disabled.
3. Interpret the social context of a given design and its implementation.	Label the social context of a given design and its implementation.	Interpret the social context of a given design and its implementation.	Analyze the social context of a given design and its implementation.
4. Describe the impact of the underrepresentation of diverse populations in the computing profession (e.g., industry culture, product diversity).	Identify how the underrepresentation of diverse populations impacts the computing profession (e.g., industry culture, product diversity).	Describe the impact of the underrepresentation of diverse populations in the computing profession (e.g., industry culture, product diversity).	Assess the impact of the underrepresentation of diverse populations in the computing profession (e.g., industry culture, product diversity).
SP/Analytical Tools Knowledge Unit			
1. Analyze stakeholder positions in a given situation.	Define stakeholder positions in a given situation.	Analyze stakeholder positions in a given situation.	Formulate stakeholder positions in a given situation.
2. Discuss ethical/social tradeoffs in technical decisions.	Identify ethical/social tradeoffs in technical	Discuss ethical/social tradeoffs in technical	Evaluate ethical/social tradeoffs in technical

	decisions.	decisions.	decisions.
3. Describe user responsibilities related to the handling of information in both personal and enterprise computing.	Define types of information handled in both personal and enterprise computing.	Describe responsibilities related to the handling of information in both personal and enterprise computing.	Analyze the effects of improper handling of information in both personal and enterprise computing.
4. Describe potential cyber attacks and the actors that might perform them.	Identify potential cyber attacks and the actors that might perform them.	Describe potential cyber attacks and the actors that might perform them.	Analyze the impact of potential cyber attacks and the likelihood of the attacks happening.
SP/Professional Ethics Knowledge Unit			
1. Discuss various types of ethical issues and dilemmas in both personal and enterprise computing.	Identify various types of ethical issues and dilemmas in both personal and enterprise computing.	Discuss various types of ethical issues and dilemmas in both personal and enterprise computing.	Debate various types of ethical issues and dilemmas in both personal and enterprise computing.
2. Explain recent and historical legislation related to digital privacy, unlawful access, and digital piracy, cyber defense, and computing ethics.	Identify recent and historical legislation related to digital privacy, unlawful access, digital piracy, cyber defense, and computing ethics.	Explain recent and historical legislation related to digital privacy, unlawful access, and digital piracy, cyber defense, and computing ethics.	Assess the consequences of violating any of the legislation related to digital privacy, unlawful access, digital piracy, cyber defense, or computing ethics.
3. Analyze the impact of Acceptable Use Policies and Online Codes of Conduct on employee behavior and choices in the digital space.	Identify the most commonly listed items in an Acceptable Use Policy and an Online Code of Conduct as they relate to the use of digital resources.	Analyze the impact of Acceptable Use Policies and Online Codes of Conduct on employee behavior and choices in the digital space.	Appraise the Acceptable Use Policy and Online Code of Conduct statements at your school for their relevance to the current period and technology.
4. Summarize case studies related to ethics.	Identify components associated with ethics in two case studies related to ethics in computing.	Summarize the long- and short-term impacts of the actions listed in the two case studies.	Debate, in both face-to-face and online format, the main actions listed in the case studies and the subsequent impact of these actions.
SP/Intellectual Property Knowledge Unit			
1. Explain the terms intellectual property, fair-use, copyright, and plagiarism.	Define the terms intellectual property, fair-use, copyright, and plagiarism. Give examples of each.	Explain, with examples, the terms intellectual property, fair-use, copyright, and plagiarism. Discuss the	Debate ethics violations as related to copyright or the fair-use doctrine.

	State the plagiarism policy at your school.	importance of knowing these terms in the current age of the world wide web.	
2. Discuss the key pieces of legislation related to fair-use, plagiarism, and intellectual property copyrights.	Identify the key pieces of legislation related to fair-use, plagiarism, and intellectual property copyrights.	Discuss the key pieces of legislation related to fair-use, plagiarism, and intellectual property copyrights.	Debate the effectiveness of legislation related to the fair-use doctrine, plagiarism, intellectual property copyright protections in the global marketplace and economy.
3. Summarize laws, both national and international, related to code patents and intellectual property copyrights.	List the process of obtaining a software/ hardware patent and the laws that protect the patent.	Summarize, at length, the process of obtaining a software/ hardware patent and the laws that protect the patent.	Discuss, at length, the process of obtaining a software/hardware patent and the laws that protect the patent by referencing key cases such as Apple v Samsung.
SP/Privacy and Civil Liberties Knowledge Unit			
1. Apply solutions to privacy threats in transactional databases and data warehouses.	Identify solutions to privacy threats in transactional databases and data warehouses.	Apply solutions to privacy threats in transactional databases and data warehouses.	Evaluate solutions to privacy threats in transactional databases and data warehouses.
2. Discuss the role of data collection in the implementation of pervasive surveillance systems (e.g., RFID, face recognition, mobile computing).	Identify the types of data collected in the implementation of pervasive surveillance systems (e.g., RFID, face recognition, mobile computing).	Discuss the role of data collection in the implementation of pervasive surveillance systems (e.g., RFID, face recognition, mobile computing).	Assess the role of data collection in the implementation of pervasive surveillance systems (e.g., RFID, face recognition, mobile computing).
3. Investigate the impact of technological solutions to privacy problems.	Discuss the impact of technological solutions to privacy problems.	Investigate the impact of technological solutions to privacy problems.	Debate the impact of technological solutions to privacy problems.
SP/Professional Communication Knowledge Unit			
1. Demonstrate competency in oral, written, and visual communication in the computing profession.	Identify a readable, concise, and effective paper on a given technical topic using proper citation methods.	Demonstrate communication skills by writing a thorough review/research paper using proper citations with graphics incorporated into the	Debate/Discuss the technical topic at hand. Arguments should be thorough, clearly articulated, and specifically tuned to a particular audiences.

		paper. Accurate use of language and grammar is important.	
2. Distinguish between verbal and nonverbal communication.	List the various ways in which nonverbal cues can impact the message being delivered.	Distinguish the importance of both verbal and nonverbal techniques in an oral presentation. Cite proper sources for this paper.	Evaluate the various ways in which nonverbal cues can impact the message being delivered.
3. Demonstrate a broad grasp of communication theories as they apply to communication in the world of technology.	Identify the communication theories as they apply to technical writing and technical communication.	Demonstrate established communication theories as they apply to technical writing and technical communication.	Choose any one communication theory and write a paper to explain it in depth. Include in your explanation a discussion of how the theory applies to written, oral, and nonverbal communication in the world of technology.
SP/Sustainability Knowledge Unit			
1. Describe the economic, social, and environmental impacts of computing.	List the various ways in which computing or use of e-resource impacts the economy, the society, and the environment around us.	Describe the various ways in which computing or use of e-resource impacts the economy, the society, and the environment around us.	Debate processes related to the various ways in which computing or use of e-resource impacts the economy, the society, and the environment around us. An example of a debatable topic could be Reverse Supply Chains.
2. Discuss strategies used to assess and lessen the carbon footprint of materials and equipment used in computing.	Find at least three news item/scholarly articles related to the environmental footprint of the manufacture and use of computers.	Discuss at least three news item/scholarly articles related to the environmental footprint of the manufacture and use of computers.	Debate at least three news item/scholarly articles related to the environmental footprint of the manufacture and use of computers.
3. Summarize case studies related to sustainable computing efforts.	Identify sustainable computing efforts in a case study.	Summarize a case study related to sustainable computing and respond to summary questions at the end of the case study.	Develop solutions for the problems listed in a case study related to sustainable computing and respond to summary questions at the end of the case study.

SP/Security Policies, Laws and Computer Crime Knowledge Unit			
1. List examples of computer crimes and social engineering incidents with societal impact.	List examples of computer crimes and social engineering incidents with societal impact.	Summarize examples of computer crimes and social engineering incidents with societal impact.	Investigate recent examples of computer crimes and social engineering incidents with societal impact.
2. Interpret laws that apply to computer crimes.	Identify laws that apply to computer crimes.	Interpret laws that apply to computer crimes.	Deconstruct laws that apply to computer crimes.
3. Describe the motivation and ramifications of cyber terrorism and criminal hacking.	Recognize the motivation and ramifications of cyber terrorism and criminal hacking.	Describe the motivation and ramifications of cyber terrorism and criminal hacking.	Evaluate the motivation and ramifications of cyber terrorism and criminal hacking.
4. Examine the ethical and legal issues surrounding the misuse of access and various breaches in security.	Discuss the ethical and legal issues surrounding the misuse of access and various breaches in security.	Examine the ethical and legal issues surrounding the misuse of access and various breaches in security.	Analyze the ethical and legal issues surrounding the misuse of access and various breaches in security.
5. Write a company-wide policy, which includes procedures for managing passwords, avoiding social engineering attacks, and employee monitoring.	Find policies that include procedures for managing passwords, avoiding social engineering attacks, and employee monitoring.	Write a company-wide policy, which includes procedures for managing passwords, avoiding social engineering attacks, and employee monitoring.	Compare several company-wide policies, which includes procedures for managing passwords, avoiding social engineering attacks, and employee monitoring.

Bloom's Revised Taxonomy

The foundational Taxonomy of Educational Objectives: A Classification of Educational Goals was established in 1956 by Dr. Benjamin Bloom, an educational psychologist, and is often referred to as Bloom's Taxonomy. This classification divides educational objectives into three learning domains: Cognitive (knowledge), Affective (attitude) and Psychomotor (skills). In 2000, Lorin Anderson and David Krathwohl updated Bloom's seminal framework to create **Bloom's Revised Taxonomy**, focusing on the Cognitive and Affective Domains. As described below, the ACM Committee for Computing Education in Community Colleges (CCECC) has adopted **Bloom's Revised Taxonomy** for the assessment of student learning outcomes in its computing curricula.

Cognitive Domain Action Verbs

Remembering	***Understanding***	***Applying***	***Analyzing***	***Evaluating***	***Creating***
Define	*Classify*	*Apply*	*Analyze*	*Appraise*	*Assemble*
Duplicate	*Convert*	*Calculate*	*Attribute*	*Argue*	*Construct*
Find	*Demonstrate*	*Carry out*	*Categorize*	*Assess*	*Create*
Identify	*Describe*	*Edit*	*Compare*	*Choose*	*Design*
Label	*Differentiate*	*Diagram*	*Contrast*	*Critique*	*Develop*
List	*Discuss*	*Execute*	*Decompose*	*Debate*	*Devise*
Locate	*Exemplify*	*Illustrate*	*Deconstruct*	*Defend*	*Formulate*
Memorize	*Explain*	*Implement*	*Deduce*	*Estimate*	*Hypothesize*
Name	*Infer*	*Investigate*	*Discriminate*	*Evaluate*	*Invent*
Recall	*Interpret*	*Manipulate*	*Distinguish*	*Judge*	*Make*
Recognize	*Paraphrase*	*Modify*	*Examine*	*Justify*	*Plan*
Retrieve	*Report*	*Operate*	*Integrate*	*Support*	
Select	*Summarize*	*Perform*	*Organize*	*Test*	
State	*Translate*	*Produce*	*Outline*	*Value*	
		Solve	*Structure*	*Verify*	
		Use			
		Write			

Glossary of Terms

The ACM CCCECC defines the following terms in relationship to curricula associated with computing education in associate-degree granting institutions.

Associate Degrees are well-defined and meaningful completion points at the conclusion of two-year degree programs; such degrees are awarded by two-year, community or technical colleges, as well as some four-year colleges.

Career Programs are specifically designed to enable students to pursue entry into the workforce after two years of college studies; these are typically Associate of Applied Science (AAS) degree programs.

Computing is now recognized by the ACM as comprised of six defined sub-disciplines: computer science, computer engineering, software engineering, information systems, information technology, and cybersecurity.

Computer Engineering … involves the design and construction of processor-based systems comprised of hardware, software, and communications components. This four-year curriculum focuses on the synthesis of electrical engineering and computer science as applied to the design of systems such as cellular communications, consumer electronics, medical imaging and devices, alarm systems and military technologies. Upon graduation, students initiating careers as computer engineers should be able to design and implement systems that involve the integration of software and hardware devices.

Computer Science … involves design and innovation developed from computing principles. This four-year curriculum focuses on the theoretical foundations of computing, algorithms, and programming techniques, as applied to operating systems, artificial intelligence, informatics and the like. Upon graduation, students initiating careers as computer scientists should be prepared to work in a broad range of positions involving tasks from theoretical work to software development.

Cybersecurity … The ACM JTF defines cybersecurity as a "computing-based discipline involving technology, people, information, and processes to enable assured operations. It involves the creation, operation, analysis, and testing of secure computer systems. It is an interdisciplinary course of study, including aspects of law, policy, human factors, ethics, and risk management often in the context of adversaries." (December 7, 2015, www.csec2017.org/)

Information Systems … involves the application of computing principles to business processes, bridging the technical and management fields. This four-year curriculum focuses on the design, implementation and testing of information systems as applied to business processes such as payroll, human resources, corporate databases, data warehousing and mining, e-commerce, finance, customer relations management, transaction processing, and data-driven decision making and executive support. Upon graduation, students initiating

careers as information systems specialists should be able to analyze information requirements and business processes and be able specify and design systems that are aligned with organizational goals.

Information Technology … involves the design, implementation and maintenance of technology solutions and support for users of such systems. This four-year curriculum focuses on crafting hardware and software solutions as applied to networks, security, client-server and mobile computing, web applications, multimedia resources, communications systems, and the planning and management of the technology lifecycle. Upon graduation, students initiating careers as information technology professionals should be able to work effectively at planning, implementation, configuration, and maintenance of an organization's computing infrastructure.

Software Engineering … involves the design, development and testing of large, complex, and safety-critical software applications. This four-year curriculum focuses on the integration of computer science principles with engineering practices as applied to constructing software systems for avionics, healthcare applications, cryptography, traffic control, meteorological systems and the like. Upon graduation, students initiating careers as software engineers should be able to properly perform and manage activities at every stage of the life cycle of large-scale software systems.

Transfer Programs are specifically designed for students intending to matriculate into the junior year of a four-year program; these are typically Associate of Arts (AA) or Associate of Science (AS) degree programs.

Bibliography

ACM Code of Ethics and Professional Conduct, (1992), Available from www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct

ACM/IEEE-CS Joint Curriculum Task Force, *Computer Science 2013: Curriculum Guidelines for Undergraduate Programs in Computer Science*, (2013), Available from www.cs2013.org.

ACM/IEEE-CS Joint Curriculum Task Force, Computing Curricula 2001: Computer Science, (2001).

ACM/IEEE-CS Joint Curriculum Task Force, Computing Curricula 1991, ACM Press and IEEE Computer Society Press (1991).

ACM Joint Task Force, Undergraduate Cybersecurity Curricular Guidelines (forthcoming 2017), Available from www.csec2017.org.

ACM Two-Year College Computing Curricula Task Force, Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science, ACM Press (2009).

ACM Two-Year College Computing Curricula Task Force, Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science, ACM Press (2003).

ACM Two-Year College Computing Curricula Task Force, Computing Curricula Guidelines for Associate-Degree Programs: Computing Sciences. ACM Press (1993).

American Association of Community Colleges, http://www.aacc.nche.edu/

Anderson, L.W., and Krathwohl, D.A., *A Taxonomy for Learning, Teaching, and Assessing: A Revision to Bloom's Taxonomy of Educational Objectives*, (2001).

Bloom, Benjamin S., The Taxonomy of Educational Objectives: Classification of Educational Goals. Handbook I: The Cognitive Domain, McKay Press, New York (1956).

IEEE Computer Society, www.computer.org/education/

National Institute of Standards and Technology, National Initiative for Cybersecurity Educaiton (NICE), *The National Cybersecurity Workforce Framework*, (April 2014), Available from http://csrc.nist.gov/nice/framework/

The White House, Office of the Press Secretary, *FACT SHEET: National Cybersecurity Action Plan* (NCAP), (February 2016), Available from www.whitehouse.gov/the-press-office/2016/02/09/fact-sheet-cybersecurity-national-action-plan