acm CCECC

Committee for Computing Education in Community Colleges

Computer Science Curricular Guidance

for Associate-Degree Transfer Programs

with Infused Cybersecurity

June 2017

Computer Science Transfer Curriculum 2017

Computer Science Curricular Guidance for
Associate-Degree Transfer Programs
with Infused Cybersecurity

June 30, 2017

The Association for Computing Machinery (ACM) Committee
for Computing Education in Community Colleges (CCECC)

ISBN: 978-1-4503-5369-4
DOI: 10.1145/3108241
Web link: http://dx.doi.org/10.1145/3108241

Sponsoring Society

This report was made possible with financial support
from the Association for Computing Machinery (ACM)

The CSTransfer2017 Final Report has been endorsed by
Association for Computing Machinery (ACM)
ACM Committee for Computing Education in Community Colleges (CCECC)

Cover art by Linda Wettengel
Printed in the United States

Computer Science Curricular Guidance for Associate-Degree Transfer Programs with Infused Cybersecurity

Final Report

June 2017

Produced by the members of the ACM Committee for Computing Education in Community Colleges:

Elizabeth K. Hawthorne, Ph.D., Union County College, NJ
Cara Tang, Ph.D., Portland Community College, OR
Cindy S. Tucker, Bluegrass Community and Technical College, KY
Christian Servin, Ph.D., El Paso Community College, TX
Teresa T. Moore, Volunteer State Community College, TN

Table of Contents

Figures

Tables

Acknowledgements

The members of the ACM Committee for Computing Education in Community Colleges (CCECC) acknowledge and thank the ACM Education Board for providing funding for this important project to develop computer science curricular guidance for associate-degree transfer programs with infused cybersecurity. The CCECC also sincerely thanks each of the team leaders, task force members, and other contributors (listed below) for his/her time, talent, and dedication toward making this project a success for the entire computing education community.

ACM CCECC Members	Team Leaders
Dr. Elizabeth K. Hawthorne, Union County College, NJ	Prof. Lambros Piskopos, Wilbur Wright College, IL
Dr. Cara Tang, Portland Community College, OR	Dr. Christian Servin, El Paso Community College, TX
Prof. Cindy S. Tucker, Bluegrass Community and Technical College, KY	Prof. Teresa T. Moore, Volunteer State Community College, TN
Dr. Christian Servin, El Paso Community College, TX	
Prof. Teresa T. Moore, Volunteer State Community College, TN	

Task Force Members	Other Contributors
Dr. Markus Geissler, Cosumnes River College, CA	Prof. Bryce Barrie, Saskatchewan Polytechnic, Canada
Dr. Anne Applin, Southern Maine Community College, ME	Prof. Michael Bauer, Leeward Community College, HI
Prof. Kimberly Bertschy, Northwest Arkansas Community College, AR	Prof. Paul Dadosky, Ivy Tech Community College, IN
Prof. Colleen Case, Schoolcraft College, MI	Prof. Andrea DeMott, Ohio University, OH
Prof. Rafael Escalante, El Paso Community College, TX	Dean Jamie Edwards, Wytheville Community College, VA
Dr. Becky Grasser, Lakeland Community College, OH	Dr. Larry Forman, San Diego City College, CA
Prof. Charles Hardnett, Gwinnett Technical College, GA	Prof. Guy Garrett, Gulf Coast State College, FL
Prof. Amardeep Kahlon, Austin Community College District, TX	Prof. Dianne Hill, Jackson College, MI
Prof. James Kolasa, Bluegrass Community and Technical College, KY	Dr. Nancy Jones, Coastline Community College, CA
Dr. Shamsi Moussavi, MassBay Community College, MA	Prof. Marc Nester, Wytheville Community College, VA
Prof. Pam Schmelz, Ivy Tech Community College, IN	Dr. Dean Nevins, Santa Barbara City College, CA
Prof. Melissa Stange, Lord Fairfax Community College, VA	Dr. Michael Posner, Villanova University, PA
Prof. Khallai Taylor, Miami-Dade College, FL	Prof. Kristopher Roberts, Ivy Tech Community College, IN
Prof. Carole Tharnish, Northeast Community College, NB	Prof. Barry Sullens, Ivy Tech Community College, IN
	Prof. Robert Surton, Columbia Gorge Community College, OR

Introduction

Overview of the Curricular Development Process

During SIGCSE 2017, the ACM CCECC unveiled the last draft version to receive any final feedback from the broader computing education community. The CCECC engaged the community through a pre-symposium event, Birds-of-a-Feather and Poster sessions, as well as individuals stopping by either the Community College Reception or ACM booth in the Exhibit Hall to talk with members of the CCECC about this curricular guidance.

Starting with the ACM/IEEE-CS CS2013 Curricular Guidelines (ACM and IEEE-CS, 2013), the members of the CCECC followed an iterative curriculum revision and development process to update the ACM 2009 *Guidelines for Associate-Degree Transfer Curriculum in Computer Science*, which produced this 2017 curricular guidance, *Computer Science Curricular Guidance for Associate-Degree Transfer Programs with Infused Cybersecurity*. Two draft versions were released for public review and comment. Community feedback and survey results from the first comment period for the initial draft, *StrawDog*, were processed to produce the second draft, *IronDog*. Similarly, feedback on *IronDog* was used to produce the final draft presented at SIGCSE2017. Throughout the two-year process, over 70 community college and university computing educators contributed to the creation of this final version 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, stopping by the poster session at ITiCSE 2016 in Arequipa, Peru, or attending one of several sessions at SIGCSE 2017 in Seattle, WA.

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 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.

As with most countries, cybersecurity is a national priority in the United States with a formal action plan (The White House, 2016). Higher education is a key component of the cybersecurity national action plan. 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 Engineers (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 for cybersecurity education (ACM/IEEE/AIS/IFIP Joint Task Force, 2017).

In recognition of the national importance of cybersecurity education, the ACM CCECC carefully wove contemporary cybersecurity throughout its Computer Science Transfer Curriculum. See Appendix A for a consolidated list of all the cybersecurity-related student learning outcomes woven into the Computer Science Transfer Curriculum. The cybersecurity concepts of the curriculum were informed by the draft version of the ACM/IEEE/AIS/IFIP *Cybersecurity Curricular 2017: Curriculum Guidelines for Undergraduate Degree Programs in Cybersecurity* (ACM/IEEE/AIS/IFIP Joint Task Force, 2017), the NICE National Cybersecurity Workforce

Framework (National Institute of Standards and Technology, 2016), and the knowledge units of the NSA/DHS National Centers of Academic Excellence in Cyber Defense for Two-Year Colleges (CAE2Y) (NSA and Department of Homeland Security, 2017).

Survey Input

Two surveys provided valuable input influencing this curricular guidance. The first survey asked which knowledge areas and knowledge units from CS2013 are appropriate in the first two years of a computer science program. A second survey solicited input on cybersecurity content appropriate in a computer science program in the first two years. The surveys were disseminated internationally and received approximately 50 responses from eight different countries. Responses to the first survey indicated that 17 of the 18 knowledge areas of CS2013 are appropriate, in some part and at some level, in the first two years of a computer science transfer program.

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” (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:

- ❖ transfer into baccalaureate programs;
- ❖ entrance into the local workforce; and
- ❖ 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 and 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, as well as 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 specific 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, whether designed specifically to enable graduates to transfer into the upper division of a baccalaureate program or to gain entry into the workforce. Additionally, 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 programs (typically A.A.S.) provide the specific knowledge, skills, and abilities (KSA) necessary to proceed directly into the workplace, while transfer-oriented degree programs (typically A.S.) provide the academic foundation and pathway to continue a program of study at a four-year college or university.

Diversity in the Computing Profession

Across the globe there is a high demand for computing professionals and a significant shortfall in satisfying job vacancies in many locations. In the U.S. alone, it is anticipated that the current graduation rates in computing disciplines may satisfy only one-third of the projected 1.4 million computing-related jobs openings in the coming years. Worldwide, the growth of new and emerging roles in computer, technology, and engineering fields exceeds the rate that underrepresented groups enter these fields. Academic research continues to bear light on the pressing need to increase the diversity of students pursuing computer science degrees and the numerous benefits of doing so. To help fulfill the increasing shortage of computer professionals, computer science faculty should increase efforts to effectively recruit and retain a wider range of students and build and provide effective support structures so that all students can successfully graduate. See Glossary of Terms.

Cybersecurity in Computing Curricula

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 cybersecurity challenges they will inevitably encounter in their careers as computing professionals. This can be addressed through a variety of implementation strategies. One approach that some associate-degree computing programs offer is a number of individual courses on specific security topics. This approach can provide many content opportunities for specialization but may leave students who don't choose the specialized security courses without 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. The guidelines presented here 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. See **Appendix A** for a consolidated list of all the cybersecurity-related student learning outcomes organized by knowledge area and unit.

Ethics and Professionalism

Ethical reasoning and professional conduct are important concepts in the overall curricula for computing disciplines including computer science, and must be integrated throughout the programs of study. This ethical and professional context should be established at the onset and 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." The Code goes on to provide an excellent framework for conduct that should be fostered beginning early in students' experiences. *(www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct)*.

As computing technologies become ubiquitous in society, ethical behavior and adherence to codes of conduct for computing professionals are 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.

Characteristics of Computer Science Graduates

Graduates of associate-degree programs in computer science should possess foundational competency in the areas described throughout the Body of Knowledge (BoK) presented in these curriculum guidelines. Increasingly, the area of computing has become critical to the operation of many organizations. Computing employees must demonstrate professionalism and ethical behavior (as described above), 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. The world of work is changing at an increasingly rapid pace and employers are seeking graduates who are adaptable and flexible in the workplace. Computer science graduates must easily, and quickly, adjust to mutable circumstances and environments, embrace new ideas, and demonstrate resourcefulness while still adding benefit to the organization. This field also demands that professionals engage in lifelong learning, an ongoing process of professional growth and development, to ensure that their skills and abilities remain current with ever-changing technology.

To this workplace readiness goal, faculty are strongly encouraged to incorporate professional practices and applied work as an integral part of computer science programs to benefit students. Students should be encouraged to:

- work in teams;
- use techniques of task and time management;
- solve practical problems in course projects;
- solve problems which demonstrate creativity, adaptability, and flexibility;
- make oral presentations;
- communicate effectively in writing;
- confront issues of privacy, confidentiality and ethics;
- use current technology in computer laboratories;
- attain real-world experience through cooperative education, internships, and/or other practicum activities; and
- participate in student chapters of computing societies and organizations (e.g., ACM) for professional development opportunities.

Internationalization

In the process of developing its curricular guidance in computer science, the ACM CCECC continually sought international perspectives from two-year post-secondary ("tertiary") higher education programs both locally within the United States and globally throughout the world. Feedback was received and processed from countries around the globe, including China, India, Turkey, South Africa, Australia, New Zealand, Canada, Mexico, Peru, Brazil, the United Kingdom, and countries across the European Union.

Through their Office of International Programs and Services, the American Association of Community Colleges "promotes partnerships between U.S. community colleges and institutions around the world to support the exchange of best practices, enhance mutual understanding between cultures, and expand opportunities for students to gain global competence and skills for the 21st Century." Current international partnerships include China, Indonesia, India, Brazil, and Mexico (American Association of Community Colleges, 2017). Other countries recognize the economic benefit of graduates with two-year degrees, including Canada and China, for example. The Association of Canadian Community Colleges offers a wealth of information on two-year degree programs and colleges in Canada (Association of Canadian Community Colleges, 2017). The Ministry of Education of the People's Republic of China describes institutions like community colleges in its discussion on two- to three-year higher vocational education with the emphasis on high-level professional technical talents (Ministry of Education of the People’s Republic of China, 2017).

Assessment

The cognitive outcomes in this guidance are clustered by related Knowledge Areas (KAs). The KAs are further organized 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 metric 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). It is important to note that the middle tier of “developed” indicates the student has achieved the intended level of the learning outcome.

Articulation and Pathways

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.

Transfer Programs

Typically, associate-degree computing programs fall into two categories: those designed for transfer into baccalaureate-degree programs (e.g., Associate in Science and Associate in Arts) and those designed to prepare graduates for immediate entry into career paths (e.g., 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 specific 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, 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 (A.A.S.). In addition, many students will augment their formal studies with technical industry 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 and 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; and
- 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 elect to further their education at the baccalaureate level (frequently with employer tuition assistance plans).

Computer Science Laboratory Experience

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, in 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 institutions granting associate degrees 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.

Mathematics Requirements

A strong foundation in mathematics provides the necessary basis for associate-degree transfer programs in computing. This foundation must include both mathematical techniques and formal mathematical reasoning. Mathematics provides a language for working with ideas relevant to computing, specific tools for analysis and verification, and a theoretical framework for understanding important concepts. For these reasons, mathematics content must be initiated early in the student’s academic career, reinforced frequently, and integrated into the student’s entire course of study. Curriculum content, pre- and co-requisite structures, and learning activities and laboratory assignments must be designed to reflect and support this framework. Many students enter two-year colleges with insufficient mathematics preparation for a computing program. Such students must devote additional semesters to achieve the mathematical maturity and problem-solving skills required to be successful in computing coursework.

Furthermore, computer science programs must provide students with a level of “mathematical maturity.” Lower division, undergraduate computer science programs need enough mathematical maturity to have the basis on which to build CS-specific mathematics. The concepts established in a course on *Discrete Structures* are foundational material for computer science, and for that reason such coursework must be completed early in the program of study. The transfer guidelines include an entire Knowledge Area on Discrete Structures estimated at 40 professor/student contact hours to cover the 34 recommended learning outcomes. The typical pre-requisite for a discrete structures course is pre-calculus; and a typical discrete structures course includes requisite concepts in set theory, induction, recursion, logic, graph theory, and combinatorics, and uses the notion of formal mathematical proof as a unifying theme. These concepts are critical to the study of algorithms and data structures. The recommended Discrete Structures course can be taught very successfully by computer science faculty with appropriate qualifications; in this manner, the content can be presented from the computing perspective, with examples and assessment activities tailored to that perspective as well. Concepts in discrete structures also serve as underpinnings for advanced computer science topics. For example, an ability to create and understand a formal proof is essential in formal specification, in verification, and in cryptography; professionals use graph theory concepts in networks, operating systems, and compilers and set theory concepts in software engineering and in databases.

The theoretical concepts of calculus not only help provide mathematical maturity, but also are required for studying the efficiency of algorithms, typically measured by *Big-O* notation. An introductory calculus course that includes the foundational concepts of limits, functions, and upper and lower bounds necessary for understanding asymptotic analysis will serve a computer science major well. Mathematics faculty typically teach this course, intended for engineering, science or mathematics majors. For computer science majors, the ability to think abstractly and to generate software solutions of mathematical models for real-world scenarios is enhanced through the study of calculus. Because introductory calculus is a well-established, time-honored course, the transfer guidelines do not include learning outcomes for the application of calculus to computer science.

Other types of mathematics courses may also be appropriate for undergraduate computer science majors. With the preponderance of algorithms for big data analytics, data visualization, and machine learning, lower division, undergraduate courses in linear algebra as well as probability and statistics will serve a computer science major well for emerging and unforeseen careers. This curricular guidance does not include student learning outcomes for these mathematics courses.

Laboratory Science Requirements

Rigorous laboratory science courses such as physics, chemistry, and biology provide computer science students with direct hands-on laboratory experiences and strong training in the tenets of the scientific method. The scientific method presents a structured methodology for much of the discipline of computing; it also provides a process of abstraction that is vital to developing a framework for logical thought. Learning activities and laboratory assignments found in computer science courses should be designed to incorporate and reinforce this framework of experimentation and observation. Furthermore, advisors should guide students intending to transfer into a baccalaureate program (immediately or as a long-term goal) to select specific science coursework appropriate to that objective. Program requirements of this nature can provide students with a crucial foundation should they later pursue interdisciplinary computing careers in scientific domains, such as astronomy, geography, biology, chemistry, or physics.

Student Support Services

Student support services aid students specifically through peer and professional counseling, mentoring, and tutoring. The format of these services may vary depending on the institution's mission and/or the academic program. Frequently, student support services are composed of students and peer mentor from the same institution, supervised by professional staff/faculty members. Student support services are primarily focused on student success and retention. Examples of these programs may occur through the following options.

- Tutoring Learning Centers provide academic support to students through tutoring and computer assisted instruction for various subjects. These centers provide a supportive environment to promote positive assistance to students and provide supplemental resources for coursework.
- Peer-led Team Learning Models provide active and collaborative discussion sessions among students. Usually these sessions take place in a peer-based learning environment, such as in computer labs or at a learning center facility. These sessions allow peer leaders to discuss challenging concepts through active learning strategies.
- Counseling Centers offer options in career pathways to students who are undecided in which track of computing to specialize. Normally, counselors work together with faculty in the corresponding programs to advise undecided students.

The Body of Knowledge with Assessment Metrics

Organization of the Body of Knowledge

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 17 of the 18 Knowledge Areas, purposefully excluding Intelligent Systems, since this KA consists mostly of elective content that is more appropriate for upper division undergraduate instruction. It is also important to note that a Knowledge Area does not necessary equate to a course. Learning outcomes from various KAs are often distributed among a single computer science course. For example, student learning outcomes from Software Development Fundamentals (SDF), Cybersecurity (CYB), Algorithms and Complexity (AL), Programming Languages (PL), and Operating Systems (OS) knowledge areas are often combined to form an introductory programming course.

Table 1 Knowledge Areas of the Computer Science Transfer Curriculum

Algorithms and Complexity (AL)	Architecture and Organization (AR)
Computational Science (CN)	Cybersecurity (CYB) (Information Assurance and Security, IAS, in CS2013)
Discrete Structures (DS)	Graphics and Visualization (GV)
Human-Computer Interaction (HCI)	Information Management (IM)
Networking and Communications (NC)	Operating Systems (OS)
Parallel and Distributed Computing (PD)	Platform-based Development (PBD)
Programming Languages (PL)	Software Development Fundamentals (SDF)
Software Engineering (SE)	Systems Fundamentals (SF)
Social Issues and Professional Practice (SP)	----

The 17 Knowledge Areas in Table 1 along with the associated Knowledge Units in Table 2 comprise the ACM Body of Knowledge for associate-degree transfer curriculum in computer science.

Table 2 Knowledge Units of the Computer Science Transfer Curriculum

Algorithms and Complexity KA Knowledge Units	**Architecture and Organization KA Knowledge Units**
Basic Analysis	Digital Logic and Digital Systems
Algorithmic Strategies	Machine Level Representation of Data
Fundamental Data Structures and Algorithms	Assembly Level Machine Organization
Basic Automata, Computability, and Complexity	Memory System Organization and Architecture
Computational Science KA Knowledge Units	**Cybersecurity KA Knowledge Units**
Introduction to Modeling and Simulation	Foundational Concepts in Security
	Principles of Secure Design
	Defensive Programming
	Threats and Attacks
	Cryptography
	Web Security
Discrete Structures KA Knowledge Units	**Graphics and Visualization KA Knowledge Units**
Sets, Relations, and Functions	Fundamental Concepts
Basic Logic	
Proof Techniques	
Basics of Counting	
Graphs and Trees	
Discrete Probability	
Human Computer Interaction KA Knowledge Units	**Information Management KA Knowledge Units**
Foundations	Information Management Concepts
Designing Interaction	Database Systems
	Data Modeling
Networking and Communications KA Knowledge Units	**Operating Systems KA Knowledge Units**
Introduction	Overview of Operating Systems
Networked Applications	Operating System Principles
	Concurrency
	Memory Management
	Security and Protection
	Virtual Machines

Parallel and Distributed Computing KA Knowledge Units	**Platform-based Development KA Knowledge Units**
Parallelism Fundamentals	None specified
Communication and Coordination	
Cloud Computing	
Programming Languages KA Knowledge Units	**Software Development Fundamentals KA Knowledge Units**
Object-Oriented Programming	Algorithms and Design
Functional Programming	Fundamental Programming Concepts
Event-Driven and Reactive Programming	Fundamental Data Structures
Basic Type Systems	Development Methods
Software Engineering KA Knowledge Units	**Systems Fundamentals KA Knowledge Units**
Software Processes	Computational Paradigms
Software Project Management	Cross-Layer Communications
Tools and Environments	Parallelism
Requirements Engineering	
Software Design	
Software Construction	
Software Verification and Validation	
Social Issues and Professional Practice KA Knowledge Units	
Social Context	
Analytical Tools	
Professional Ethics	
Intellectual Property	
Privacy and Civil Liberties	
Professional Communication	
Sustainability	
Security Policies, Laws, and Computer Crime	

Table 3 shows the estimated number of professor/student contact hours for each knowledge area, as well as the percentage. It is important to note that contact hours are **not** the same as course credit hours, such as a three-credit or four-credit course. Contact hours approximate the instructional time professors and students spend in teaching and learning together.

Table 3 Estimated Contact Hours and Percentage by Knowledge Area

Knowledge Area	Estimated Contact Hours	Percentage
Algorithms and Complexity	15	7.7%
Architecture and Organization	10	5.2%
Computational Science	1	0.5%
Cybersecurity	20	10.3%
Discrete Structures	40	20.6%
Graphics and Visualization	2	1.0%
Human-Computer Interaction	5	2.6%
Information Management	6	3.1%
Networking and Communications	5	2.6%
Operating Systems	10	5.2%
Parallel and Distributed Computing	3	1.5%
Platform-based Development	0	0%
Programming Languages	15	7.7%
Software Development Fundamentals	30	15.5%
Software Engineering	15	7.7%
Systems Fundamentals	5	2.6%
Social Issues and Professional Practice	12	6.2%
Estimated Total Contact Hours	**194**	**100%**

Assessment of Student Learning Outcomes

Each Knowledge Unit is complete with measurable student learning outcomes. Learning outcomes are expressed using action verbs from Bloom's Revised Taxonomy. The Bloom's level—*Remembering*, *Understanding*, *Applying*, *Analyzing*, *Evaluating*, or *Creating*—is indicated in bracketed italics *[italics]* after each learning outcome. See **Appendix B** for a table of the measurable Bloom's verbs that were used in the creation of this curricular guidance.

A three-tiered assessment rubric (Table 4) labeled as *emerging*, *developed*, and *highly developed* provides further clarity to each learning outcome. Typically, as the level of student achievement progresses from *emerging* to *highly developed*, the level of Bloom's action verb also increases from the lower order thinking skills (LOTS) to the higher order thinking skills (HOTS). **The *developed* middle tier indicates the student has met the intended learning expectation expressed by the outcome**.

Table 4 Sample Three-Tier Assessment Rubric

Knowledge Area	**Assessment Rubric**		
	Emerging	**Developed**	**Highly Developed**
Knowledge Unit			
Learning Outcome		*Learning Expectation*	

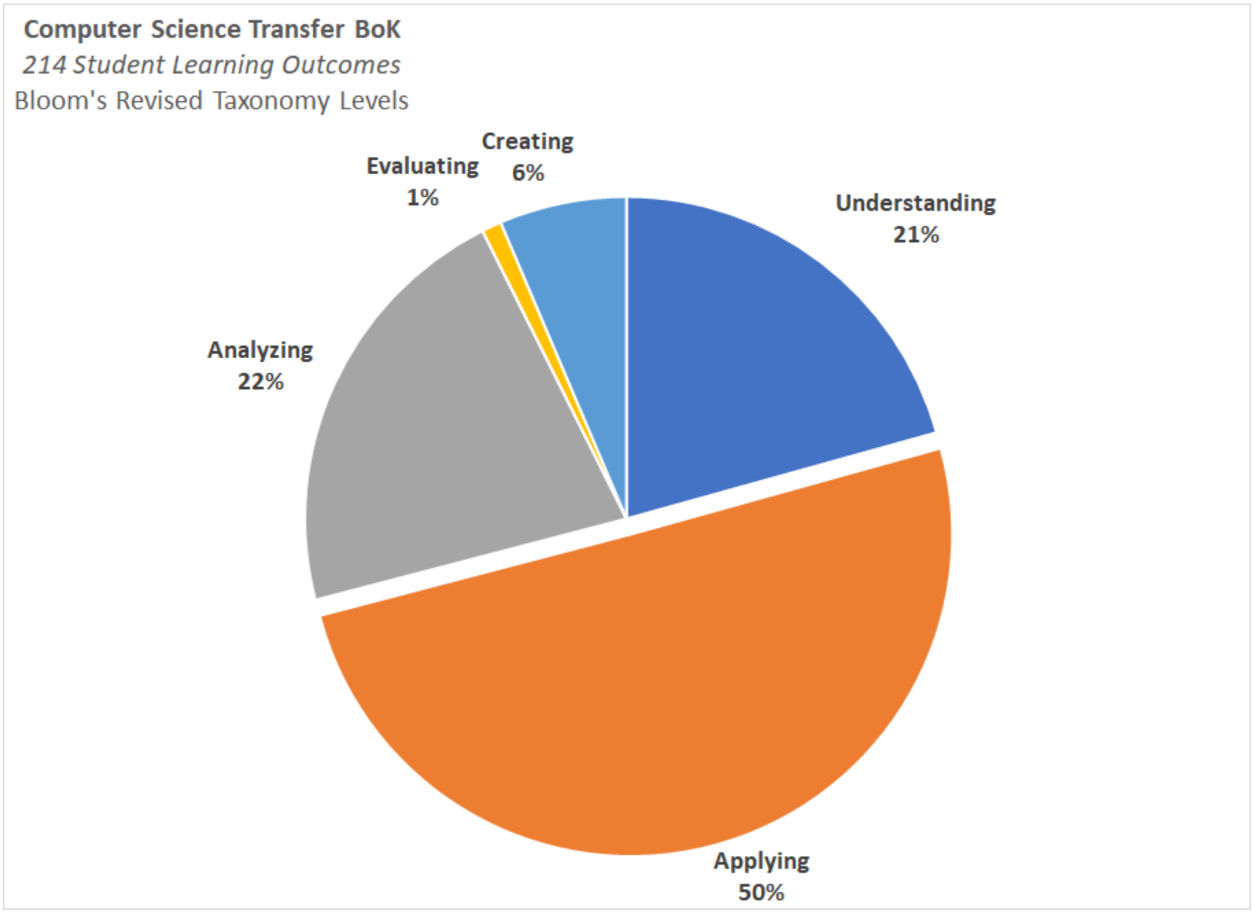

Figure 1 Pie Chart of Bloom's Revised Taxonomy Levels: All KA Learning Outcomes

Figure 1 shows the percentage of all 214 student learning outcomes in this Body of Knowledge by Bloom’s Revised Taxonomy level. Only 21% are at the *Understanding* level, meaning 79% are at the *Applying* level or higher. It is important to note that there are no middle-tier learning outcomes at the lowest Bloom’s level of *Remembering*.

Knowledge Areas

This section comprises the details of each of the 17 Knowledge Areas and associated Knowledge Units of the Computer Science Transfer Curriculum.

Algorithms and Complexity Knowledge Area (AL)

The Algorithms and Complexity (AL) knowledge area (KA) consists of 17 measurable student learning outcomes across four knowledge units. As indicated in Figure 2, most of the student learning outcomes, 53%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (18%), *Analyzing*, (23%), and *Evaluating* (6%) levels. The ACM CCECC estimates 15 professor/student contact hours for the AL knowledge area.

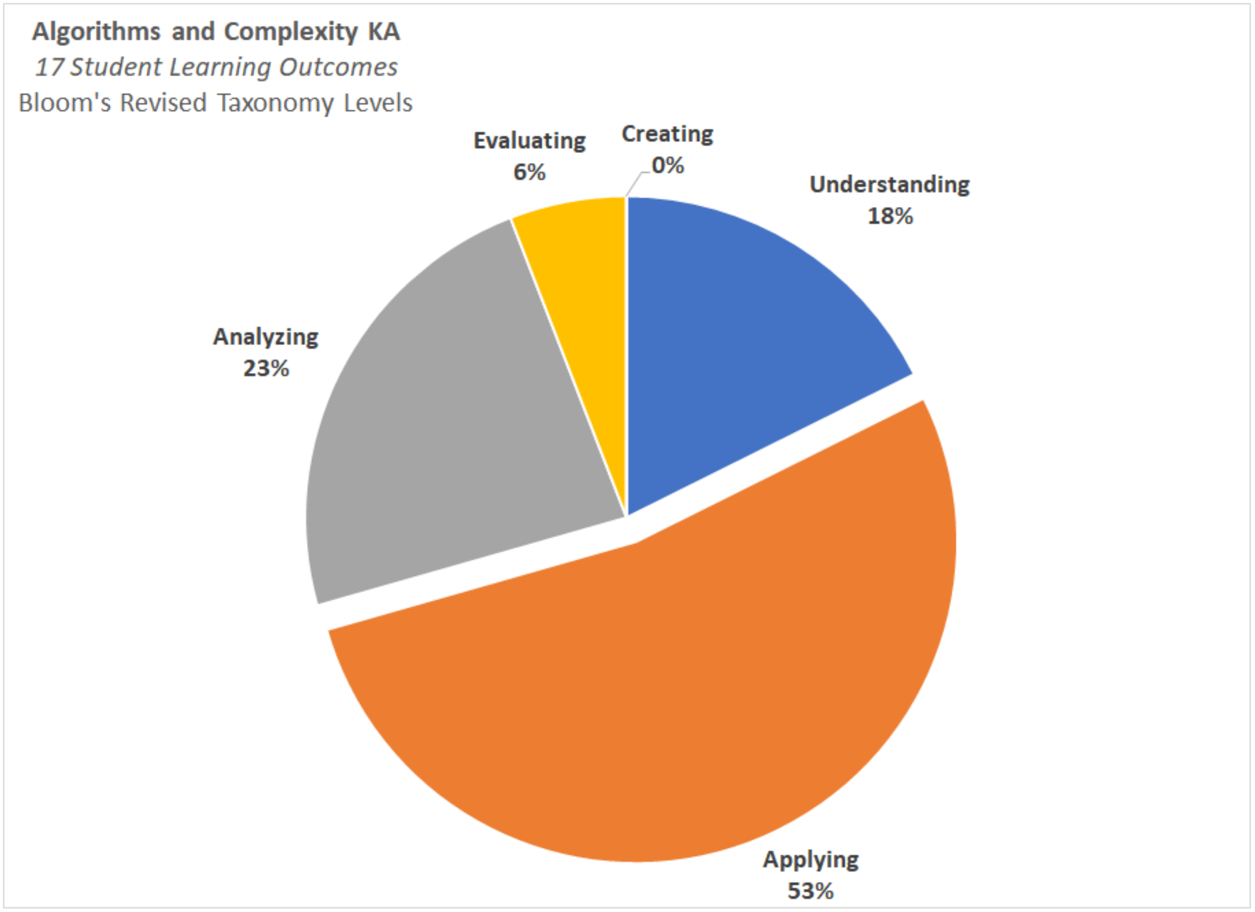

Figure 2 Pie Chart of Bloom's Revised Taxonomy Levels: AL KA LOs

The Algorithms and Complexity knowledge area defines the central concepts and skills required to design, implement, and analyze algorithms for solving problems. Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends on the algorithms chosen as well as the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect, such as security. An important part of computing is the ability to select algorithms appropriate to specific purposes and to apply them, recognizing the possibility that no suitable algorithm may exist. This facility relies on understanding the range of algorithms that address an important set of well-defined problems, recognizing their strengths and weaknesses, and their suitability in specific contexts. Efficiency is a pervasive theme throughout this knowledge area. (*adapted from CS2013, p. 55*).

Learning Outcome	**Assessment Rubric**		
AL. Algorithms and Complexity KA	**Emerging**	**Developed**	**Highly Developed**
AL/Basic Analysis Knowledge Unit			
AL-01. Analyze best, average, and worst-case behaviors of an algorithm. *[Analyzing]*	Illustrate best, average, and worst-case behaviors of an algorithm. *[Applying]*	Analyze best, average, and worst-case behaviors of an algorithm. *[Analyzing]*	Evaluate best, average, and worst-case behaviors of an algorithm. *[Evaluating]*
AL-02. Estimate time and space complexities for a given algorithm using Big-O notation. *[Evaluating]*	Distinguish between time and space complexities for a given algorithm. *[Analyzing]*	Estimate time and space complexities for a given algorithm using Big-O notation. *[Evaluating]*	Critique time and space complexities of several algorithms using Big-O and other notations. *[Evaluating]*
AL-03. Contrast standard complexity classes. *[Analyzing]*	Illustrate a few of the standard complexity classes. *[Applying]*	Contrast standard complexity classes, such as logarithmic, linear, quadratic, and exponential. *[Analyzing]*	Judge standard complexity classes as either “efficient" or "inefficient" algorithms. *[Evaluating]*
AL-04. Analyze the performances of an algorithm with various input sizes. *[Analyzing]*	Discuss the performances of an algorithm with various input sizes. *[Understanding]*	Compare the performances of an algorithm with various input sizes. *[Analyzing]*	Assess the performances of an algorithm with various input sizes. *[Evaluating]*
AL/Algorithmic Strategies Knowledge Unit (see also SDF/Algorithms and Design KU)			
AL-05. Apply an appropriate algorithmic approach to a given problem. *[Applying]*	Demonstrate an algorithmic approach to a given problem. *[Understanding]*	Apply an appropriate algorithmic approach to a given problem, such as brute-force, greedy, recursive, divide-and-conquer, and dynamic programming. *[Applying]*	Analyze the tradeoffs of various algorithmic approaches to a given problem. *[Analyzing]*

AL-06. Investigate the use of random/pseudo random number generation in cybersecurity applications. *[Applying]*	Describe the use of random numbers in cybersecurity applications. *[Understanding]*	Investigate the use of random/pseudo random number generation in cybersecurity applications, such as password generation and data encryption. *[Applying]*	Analyze the use of random/pseudo random number generation in a range of cybersecurity applications. *[Analyzing]*
AL/Fundamental Data Structures and Algorithms Knowledge Unit (see also SDF/Fundamental Data Structures KU)			
AL-07. Implement basic numerical algorithms. *[Applying]*	Describe the use of basic numerical algorithms. *[Understanding]*	Implement basic numerical algorithms, such as min, max, and mode. *[Applying]*	Develop complex numerical algorithms. *[Creating]*
AL-08. Implement common search algorithms, including linear and binary searches. *[Applying]*	Exemplify common search algorithms, including linear and binary searches. *[Understanding]*	Implement common search algorithms, including linear and binary searches. *[Applying]*	Compare the efficiency of common search algorithms, including linear and binary searches. *[Analyzing]*
AL-09. Implement common sorting algorithms, including iterative, quadratic, and recursive. *[Applying]*	Exemplify common sorting algorithms, including iterative, quadratic, and recursive. *[Understanding]*	Implement common sorting algorithms, including iterative, quadratic, and recursive. *[Applying]*	Compare the efficiency of common sorting algorithms, including iterative, quadratic, and recursive. *[Analyzing]*
AL-10. Implement hash tables, including collision avoidance and resolution. *[Applying]*	Explain the general idea of a hash table. *[Understanding]*	Implement hash tables, including collision avoidance and resolution. *[Applying]*	Compare common collision resolution techniques for hash tables. *[Analyzing]*
AL-11. Explain the runtime and memory efficiency of principal sorting, searching, and hashing functions. *[Understanding]*	Summarize the runtime and memory efficiency of either a sorting, a searching, or a hashing function. *[Understanding]*	Explain the runtime and memory efficiency of principal sorting, searching, and hashing functions. *[Understanding]*	Analyze the runtime and memory efficiency of principal sorting, searching, and hashing functions. *[Analyzing]*
AL-12. Investigate factors other than computational efficiency that influence the choice of algorithms. *[Applying]*	Describe factors other than computational efficiency that ought to be considered when choosing an algorithm. *[Understanding]*	Investigate factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the size and patterns of input data. *[Applying]*	Critique factors other than computational efficiency that influence the choice of algorithms. *[Evaluating]*

AL-13. Compare various data structures for a given problem. *[Analyzing]*	Investigate a few data structures for a given problem. *[Applying]*	Compare various data structures for a given problem, such as array, list, set, map, stack, queue, hash table, tree, and graph. *[Analyzing]*	Justify a choice of data structure for a given problem. *[Evaluating]*
AL-14. Investigate security vulnerabilities in various data structures. *[Applying]*	Summarize security vulnerabilities in various data structures. *[Understanding]*	Investigate security vulnerabilities in various data structures, such as out-of-bounds arrays and buffer overflows. *[Applying]*	Analyze security vulnerabilities in various data structures. *[Analyzing]*
AL/Basic Automata, Computability, and Complexity Knowledge Unit			
AL-15. Write a regular expression to match a pattern. *[Applying]*	Explain the use of regular expressions in pattern matching. *[Understanding]*	Write a regular expression to match a pattern. *[Applying]*	Write a regular expression to perform complex pattern matching. [Applying]
AL-16. Describe the concept of finite state machines. *[Understanding]*	Recognize a finite state machine. *[Remembering]*	Describe the concept of finite state machines. *[Understanding]*	Diagram a finite state machine. *[Applying]*
AL-17. Explain why the halting problem has no algorithmic solution. *[Understanding]*	Recognize that some problems have no algorithmic solution. *[Remembering]*	Explain why the halting problem has no algorithmic solution. *[Understanding]*	Illustrate a proof of the halting problem. *[Applying]*

Architecture and Organization Knowledge Area (AR)

The Architecture and Organization (AR) knowledge area (KA) consists of 11 measurable student learning outcomes across four knowledge units. As indicated in Figure 3, most of the student learning outcomes, 46%, are at the *Analyzing* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (9%), *Applying*, (36%), and *Creating* (9%) levels. The ACM CCECC estimates 10 professor/student contact hours for the AR knowledge area.

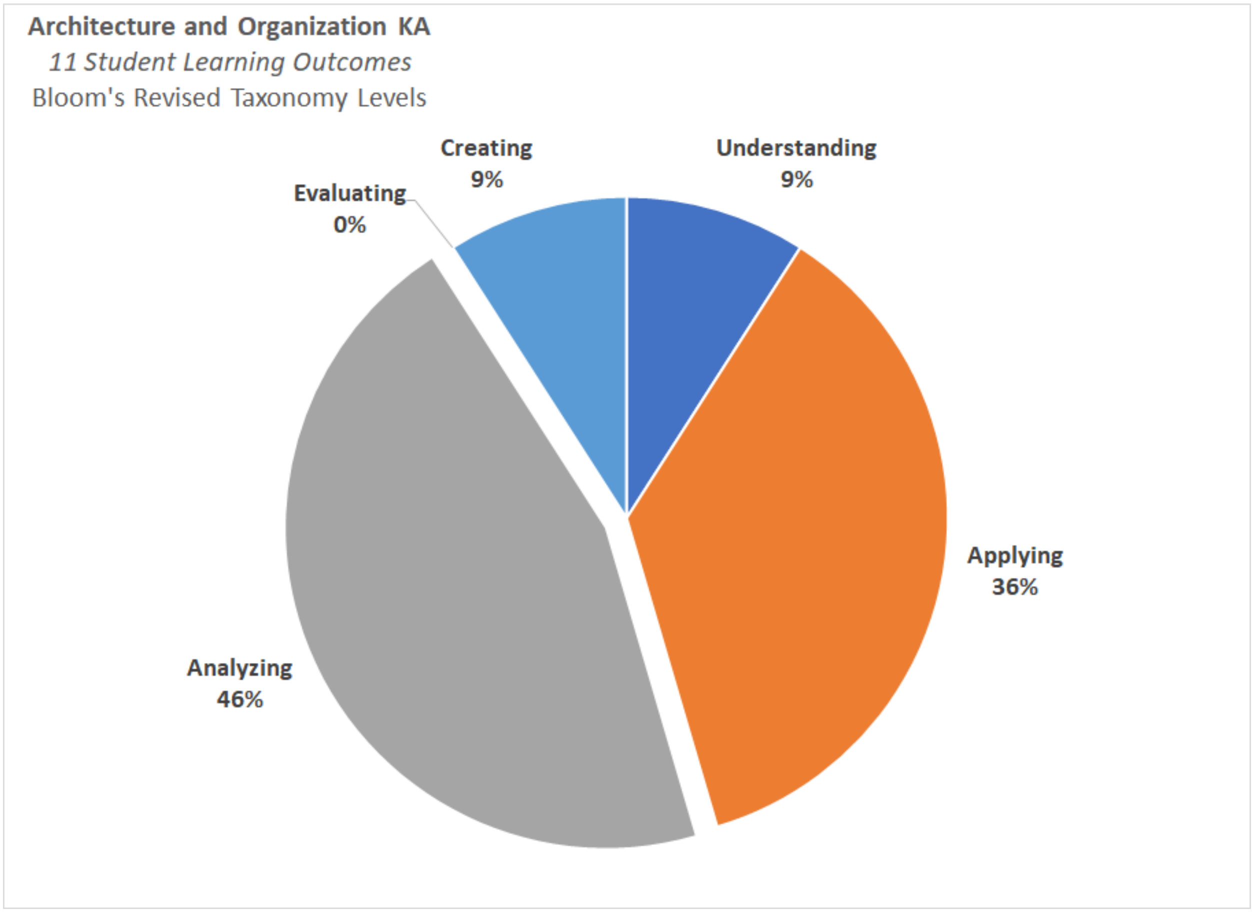

Figure 3 Pie Chart of Bloom's Revised Taxonomy Levels: AR KA LOs

Computing professionals should not regard the computer as just a black box that executes programs by magic. The knowledge area Architecture and Organization builds on the Systems Fundamentals KA to develop a deeper understanding of the hardware environment upon which all computing is based, and the interface it provides to higher software layers. Students should acquire an understanding and appreciation of a computer system's functional components, their characteristics, performance, and interactions, and, specifically, the challenge of harnessing parallelism to sustain performance improvements now and into the future. Students need to understand computer architecture to develop programs that can achieve high performance through a programmer's awareness of parallelism and latency. In selecting a system to use, students should be able to understand the tradeoff among various components, such as CPU clock speed, cycles per instruction, memory size, and average memory access time. (*adapted from CS2013, p. 62*).

Learning Outcome	**Assessment Rubric**		
AR. Architecture and Organization KA	**Emerging**	**Developed**	**Highly Developed**
AR/Digital Logic and Digital Systems Knowledge Unit			
AR-01. Diagram the digital components of computing architecture. *[Applying]*	Discuss some of the digital components of computing architecture. *[Understanding]*	Diagram the digital components of computing architecture, such as logic gates, registers, and memory. *[Applying]*	Evaluate the digital component design of a computing architecture for accuracy. *[Evaluating]*
AR/Machine Level Representation of Data Knowledge Unit			
AR-02. Analyze alternative formats to represent numerical data. *[Analyzing]*	Explain reasons for using alternative formats to represent numerical data. *[Understanding]*	Analyze alternative formats to represent numerical data. *[Analyzing]*	Choose optimal formats to represent a given set of numerical data. *[Evaluating]*
AR-03. Illustrate how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. *[Applying]*	Explain how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. *[Understanding]*	Illustrate how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. *[Applying]*	Examine how fixed-length number representations could affect accuracy and precision, causing vulnerabilities. *[Analyzing]*
AR-04. Examine the internal representation of non-numeric data. *[Analyzing]*	Illustrate internal representation of non-numeric data. *[Applying]*	Examine the internal representation of non-numeric data, such as characters, strings, records, and arrays. *[Analyzing]*	Estimate memory requirements for non-numeric data, such characters, strings, records, and arrays. *[Evaluating]*
AR-05. Compare different methods for converting numerical data from one format to another. *[Analyzing]*	Convert numerical data from one format to another. *[Understanding]*	Compare different methods for converting numerical data from one format to another, such as converting negative integers into sign-magnitude and two's-complement representations. *[Analyzing]*	Evaluate different methods for converting numerical data from one format to another. *[Evaluating]*

AR/Assembly Level Machine Organization Knowledge Unit			
AR-06. Decompose the organization and major functional units of the classical von Neumann machine. *[Analyzing]*	Diagram the organization of the classical von Neumann machine and its major functional units. *[Applying]*	Decompose the organization and major functional units of the classical von Neumann machine. *[Analyzing]*	Assess the organization of the classical von Neumann machine and its major functional units. *[Evaluating]*
AR-07. Diagram how high-level language patterns map to assembly/machine language, including subroutine calls. *[Applying]*	Summarize how high-level language patterns map to assembly/machine language, including subroutine calls. *[Understanding]*	Diagram how high-level language patterns map to assembly/machine language, including subroutine calls. *[Applying]*	Examine how high-level language patterns map to assembly/machine language, including subroutine calls. *[Analyzing]*
AR-08. Create simple assembly language program segments. *[Creating]*	Implement simple assembly language program segments. *[Applying]*	Create simple assembly language program segments. *[Creating]*	Create complex assembly language program segments. *[Creating]*
AR-09. Demonstrate the basic concepts of interrupts and I/O operations. *[Understanding]*	List basic concepts of interrupts and I/O operations. *[Remembering]*	Demonstrate the basic concepts of interrupts and I/O operations. *[Understanding]*	Implement basic concepts of interrupts and I/O operations. *[Applying]*
AR/Memory System Organization and Architecture Knowledge Unit			
AR-10. Compare the cost and performance of different types of memory technology. *[Analyzing]*	Describe different types of memory technology. *[Understanding]*	Compare the cost and performance of different types of memory technology, such as SRAM, DRAM, virtual, and cache. *[Analyzing]*	Critique the cost and performance of different types of memory technology. *[Evaluating]*
AR-11. Calculate the effect of memory latency on execution time across the memory hierarchy. *[Applying]*	Explain the effect of memory latency on execution time across the memory hierarchy. *[Understanding]*	Calculate the effect of memory latency on execution time across the memory hierarchy. *[Applying]*	Examine the effect of memory latency on execution time across the memory hierarchy. *[Analyzing]*

Computational Science Knowledge Area (CN)

The Computational Science (CN) knowledge area (KA) consists of 11 measurable student learning outcomes in one knowledge unit. As indicated in Figure 4, most of the student learning outcomes, 67%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes at the *Understanding* (33%) level. The ACM CCECC estimates one professor/student contact hour for the CN knowledge area.

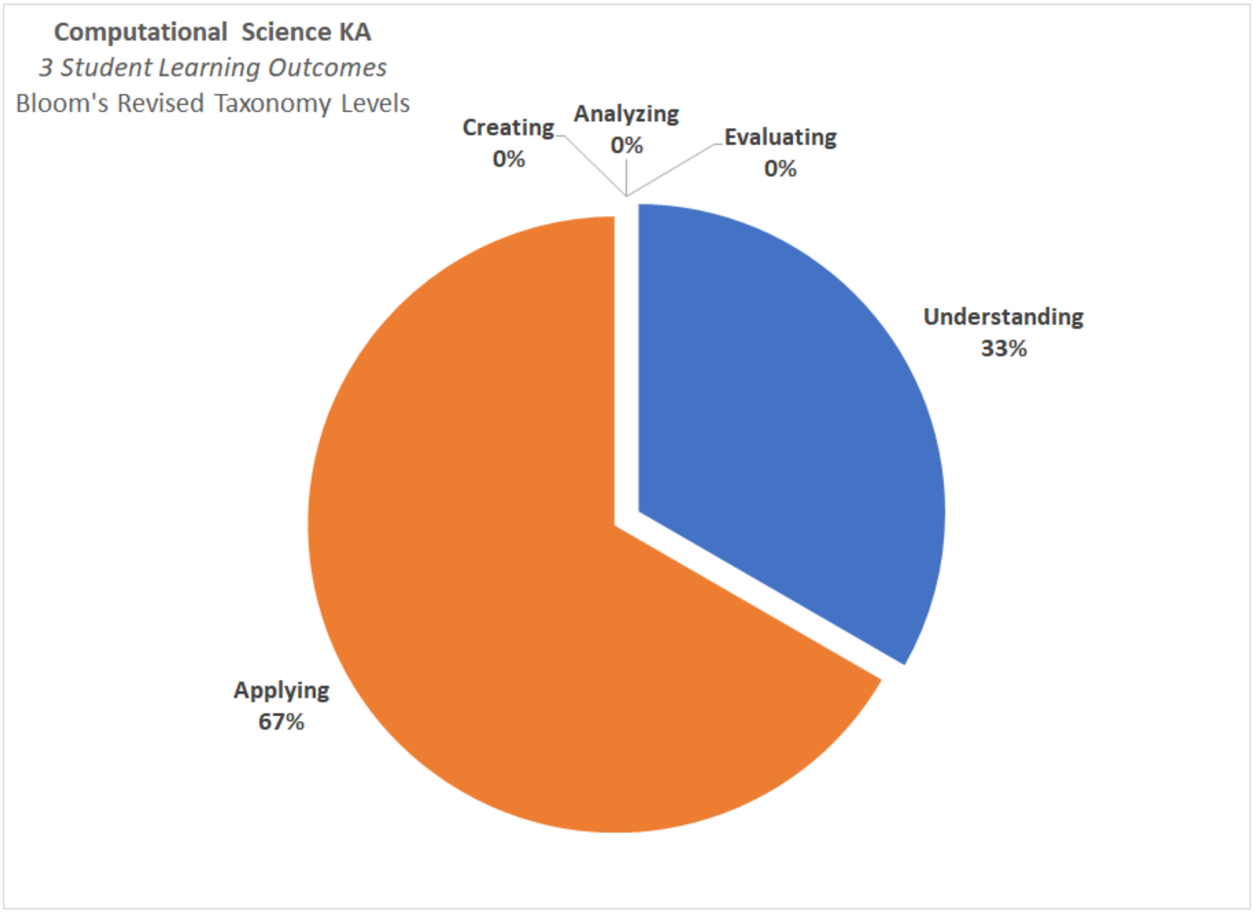

Figure 4 Pie Chart of Bloom's Revised Taxonomy Levels: CN KA LOs

Computational science is a field of applied computer science—that is, the application of computer science to solve problems across a range of disciplines, such as molecular and fluid dynamics, celestial mechanics, economics, biology, geology, medicine, and social network analysis. Fundamental concepts of computational science are germane to every computer scientist, such as modeling and simulation. This area offers exposure to many valuable ideas and techniques, including precision of numerical representation, error analysis, numerical techniques, parallel architectures and algorithms, quantum computing, modeling and simulation, information visualization, software engineering, and optimization. Topics relevant to computational science include fundamental concepts in program construction (SDF/Fundamental Programming Concepts KU), algorithm design (SDF/Algorithms and Design KU), program testing (SDF/Development Methods KU), data representations (AR/Machine Representation of Data KU), and basic computer architecture (AR/Memory System Organization and Architecture KU). (*adapted from CS2013, p. 70*).

Learning Outcome	Assessment Rubric		
CN. Computational Science KA	**Emerging**	**Developed**	**Highly Developed**
CN/Introduction to Modeling and Simulation Knowledge Unit			
CN-01. Illustrate the concepts of modeling and abstraction with respect to problem solving. *[Applying]*	Explain the concepts of modeling and abstraction with respect to problem solving. *[Understanding]*	Illustrate the concepts of modeling and abstraction with respect to problem solving. *[Applying]*	Contrast the concepts of modeling and abstraction with respect to problem solving. *[Analyzing]*
CN-02. Illustrate the relationship between modeling and simulation. *[Applying]*	Describe the relationships between modeling and simulation. *[Understanding]*	Illustrate the relationship between modeling and simulation. *[Applying]*	Examine the relationship between modeling and simulation. *[Analyzing]*
CN-03. Exemplify different types of simulations. *[Understanding]*	Identify different types of simulations. *[Remembering]*	Exemplify different types of simulations, such as physical simulations, human-guided simulations, and virtual reality. *[Understanding]*	Compare different types of simulations. *[Analyzing]*

Cybersecurity Knowledge Area (CYB) – IAS in CS2013

The Cybersecurity (CYB) knowledge area (KA) consists of 25 measurable student learning outcomes across six knowledge units. As indicated in Figure 5, most of the student learning outcomes, 57%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (21%), *Analyzing*, (17%), and *Creating* (5%) levels. The ACM CCECC estimates 20 professor/student contact hours for the CYB knowledge area.

In recognition of the national importance of cybersecurity education, the ACM CCECC carefully wove contemporary cybersecurity concepts throughout its Computer Science Transfer Curriculum. See **Appendix A** for a consolidated list of all the cybersecurity-related student learning outcomes woven throughout the Computer Science Transfer Curriculum. The Cybersecurity KA was informed by the draft version of the ACM/IEEE/AIS/IFIP *Cybersecurity Curricular 2017: Curriculum Guidelines for Undergraduate Degree Programs in Cybersecurity* (ACM/IEEE/AIS/IFIP Joint Task Force, 2017), the NICE National Cybersecurity Workforce Framework (National Institute of Standards and Technology, 2016) and the knowledge units of the NSA/DHS National Centers of Academic Excellence in Cyber Defense for Two-Year Colleges (CAE2Y) (NSA and Department of Homeland Security, 2017).

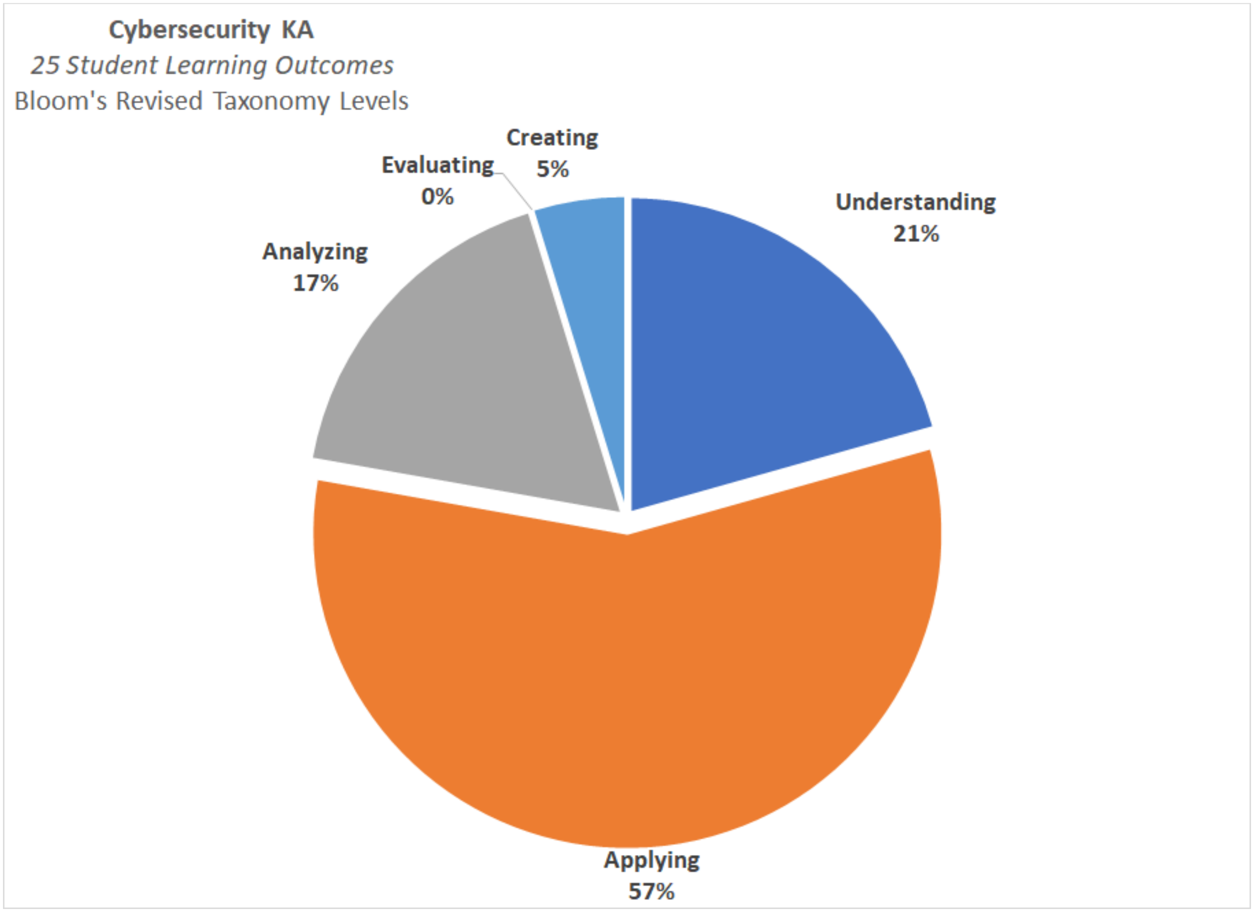

Figure 5 Pie Chart of Bloom's Revised Taxonomy Levels: CYB KA LOs

In CS2013, the Cybersecurity knowledge area is named Information Assurance and Security (IAS), and was added to the computer science Body of Knowledge in recognition of the world's reliance on technology and its critical role in computer science education. Cybersecurity is a unique Knowledge Area in this curricular guidance given the pervasiveness of the learning outcomes throughout other KAs. Cybersecurity as a domain is the set of controls and processes both technical and policy intended to protect and defend information and information systems by ensuring their confidentiality, integrity, and availability, and by providing for authentication and non-repudiation. Cybersecurity education includes all efforts to prepare students with the needed knowledge, skills, and abilities to protect our information systems and attest to the assurance of the past and current state of processes and data. The importance of security concepts and topics has emerged as a core requirement in the Computer Science discipline, much like the importance of performance concepts has been for many years. (*adapted from CS2013, p. 97*).

Learning Outcome	Assessment Rubric		
CYB. Cybersecurity KA (a.k.a. IAS in CS2013)	**Emerging**	**Developed**	**Highly Developed**
CYB/Foundational Concepts in Security Knowledge Unit			
CYB-01. Describe security as a continuous process of tradeoffs, balancing between protection mechanisms and availability. [Understanding]	Recognize the importance of security as a continuous process of tradeoffs. [Remembering]	Describe security as a continuous process of tradeoffs, balancing between protection mechanisms and availability. [Understanding]	Illustrate security as a continuous process of tradeoffs, balancing between protection mechanisms and availability. [Applying]
CYB-02. Illustrate through examples the concepts of risk, threats, vulnerabilities, attack vectors, and exploits, noting there is no such thing as perfect security. [Applying]	Differentiate the concepts of risk, threats, vulnerabilities, attack vectors, and exploits. [Understanding]	Illustrate through examples the concepts of risk, threats, vulnerabilities, attack vectors, and exploits, noting there is no such thing as perfect security. [Applying]	Compare the concepts of risk, threats, vulnerabilities, attack vectors, and exploits in a given scenario. [Analyzing]
CYB-03. Investigate various countermeasures and security controls to minimize risk and exposure. [Applying]	Explain the importance of countermeasures and security controls to minimize risk and exposure. [Understanding]	Investigate various countermeasures and security controls to minimize risk and exposure. [Applying]	Choose among various countermeasures and security controls to minimize risk and exposure in a given scenario. [Evaluating]
CYB-04. Analyze the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Analyzing]	Investigate the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Applying]	Analyze the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Analyzing]	Evaluate the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA). [Evaluating]
CYB-05. Explain the concepts of trust and trustworthiness related to cybersecurity. [Understanding]	Define the concepts of trust and trustworthiness. [Remembering]	Explain the concepts of trust and trustworthiness related to cybersecurity. [Understanding]	Diagram trust relationships in a given cybersecurity scenario. [Applying]
CYB-06. Examine ethical issues related to cybersecurity. [Analyzing]	Exemplify ethical issues in cybersecurity. [Understanding]	Examine ethical issues related to cybersecurity. [Analyzing]	Argue ethical issues related to cybersecurity. [Evaluating]

CYB-07. Illustrate various ways to minimize privacy risks and maximize anonymity. [Applying]	Describe one way to minimize privacy risks and maximize anonymity. [Understanding]	Illustrate various ways to minimize privacy risks and maximize anonymity. [Applying]	Analyze ways to minimize privacy risks and maximize anonymity in an always connected, mobile computing environment. [Analyzing]
CYB-08. Apply security principles and practices in a dynamic environment. [Applying]	Summarize security principles and practices. [Understanding]	Apply security principles and practices in a dynamic environment. [Applying]	Examine security principles and practices in a dynamic environment. [Analyzing]
CYB-09. Illustrate through examples the key role risk management frameworks play in identifying, assessing, prioritizing, and controlling risks to organizational assets. [Applying]	Paraphrase the key role risk management frameworks play in identifying, assessing, prioritizing, and controlling risks to organizational assets. [Understanding]	Illustrate through examples the key role risk management frameworks play in identifying, assessing, prioritizing, and controlling risks to organizational assets. [Applying]	Analyze a given scenario with a specific risk management framework, such as NIST, to identify, assess, prioritize, and control risks to organizational assets. [Analyzing]
CYB-10. Illustrate with examples the goals of end-to-end data security. [Applying]	Explain the goals of end-to-end data security. [Understanding]	Illustrate with examples the goals of end-to-end data security. [Applying]	Outline the goals of end-to-end data security. [Analyzing]
CYB/Principles of Secure Design Knowledge Unit			
CYB-11. Use the principles of secure design. [Applying]	Demonstrate some of the principles of secure design as related to cybersecurity. [Understanding]	Use the principles of secure design, such as least privilege, isolation, fail-safe, and deny-by-default. [Applying]	Choose appropriate secure design principles for a given cybersecurity scenario. [Evaluating]
CYB-12. Illustrate the security implications of relying on open design vs the secrecy of design. [Applying]	Discuss the security implications of relying on open design vs. the secrecy of design. [Understanding]	Illustrate the security implications of relying on open design vs. the secrecy of design. [Applying]	Analyze the security implications of relying on open design vs. the secrecy of design. [Analyzing]
CYB-13. Discuss the benefits and limitations of designing multiple layers of defenses. [Understanding]	Identify the benefits and limitations of designing multiple layers of defenses. [Remembering]	Discuss the benefits and limitations of designing multiple layers of defenses. [Understanding]	Implement multiple layers of defenses for a given scenario. [Applying]

CYB-14. Analyze the tradeoffs associated with designing security into a product. [Analyzing]	Summarize the tradeoffs associated with designing security into a product. [Understanding]	Analyze the tradeoffs associated with designing security into a product. [Analyzing]	Evaluate the tradeoffs associated with designing security into a product. [Evaluating]
CYB/Defensive Programming Knowledge Unit			
CYB-15. Construct input validation and data sanitization in applications, considering adversarial control of the input channel. [Creating] See also SDF-06.	Implement simple input validation and data sanitization in applications. [Applying]	Construct input validation and data sanitization in applications, considering adversarial control of the input channel. [Creating]	Develop complex input validation and data sanitization in applications, considering adversarial control of the input channel. [Creating]
CYB-16. Explain the tradeoffs of developing a program in a type-safe language. [Understanding] See also PL-10.	List some of the tradeoffs of developing a program in a type-safe language. [Remembering]	Explain the tradeoffs of developing a program in a type-safe language. *[Understanding]*	Compare the tradeoffs of developing a program in a type-safe language vs other types of programming languages. [Analyzing]
CYB-17. Implement programs that properly handle exceptions and error conditions. [Applying] See also SDF-12.	Explain the importance of writing programs that properly handle exceptions and error conditions. [Understanding]	Implement programs that properly handle exceptions and error conditions. [Applying]	Analyze the implementation of exception handling in programs and error conditions. [Analyzing]
CYB-18. Examine the need to update software to fix security vulnerabilities. [Analyzing]	Explain the need to update software. [Understanding]	Examine the need to update software to fix security vulnerabilities. [Analyzing]	Justify the need to update software to fix security vulnerabilities. [Evaluating]
CYB/Threats and Attacks Knowledge Unit			
CYB-19. Examine likely attack types against standalone and networked systems. *[Analyzing]*	Summarize likely attack types against software systems. *[Understanding]*	Examine likely attack types against standalone and networked systems. *[Analyzing]*	Assess likely attack types against standalone and networked systems. *[Evaluating]*
CYB-20. Illustrate the key principles of social engineering, including membership and trust. *[Applying]* **See also SP-04.**	Discuss some of the key principles of social engineering. *[Understanding]*	Illustrate the key principles of social engineering, including membership and trust. *[Applying]*	Outline the key principles of social engineering, including membership and trust. *[Analyzing]*

CYB/Cryptography Knowledge Unit			
CYB-21. Describe key terms in cryptology, including cryptography, cryptanalysis, cipher, cryptographic algorithm, and public key infrastructure. *[Understanding]*	Define some key terms in cryptology. *[Remembering]*	Describe key terms in cryptology, including cryptography, cryptanalysis, cipher, cryptographic algorithm, and public key infrastructure. *[Understanding]*	Categorize key terms in cryptology, including cryptography, cryptanalysis, cipher, and cryptographic algorithm. *[Analyzing]*
CYB-22. Use a variety of ciphers to encrypt plaintext into ciphertext. *[Applying]*	Describe basic methods for transforming plaintext into ciphertext. *[Understanding]*	Use a variety of ciphers to encrypt plaintext into ciphertext. *[Applying]*	Design a simple program to encrypt plaintext into ciphertext. *[Creating]*
CYB-23. Apply cryptographic hash functions for authentication and data integrity. *[Applying]*	Summarize the use of cryptographic hash functions for authentication and data integrity. *[Understanding]*	Apply cryptographic hash functions for authentication and data integrity. *[Applying]*	Deconstruct a cryptographic hash function used for authentication and data integrity. *[Analyzing]*
CYB-24. Contrast symmetric and asymmetric encryption in relation to securing electronic communications and transactions. *[Analyzing]*	Exemplify the difference between symmetric and asymmetric encryption. *[Understanding]*	Contrast symmetric and asymmetric encryption in relation to securing electronic communications and transactions. *[Analyzing]*	Design a security solution for a given scenario that integrates symmetric encryption with asymmetric encryption. *[Creating]*
CYB/Network Security Knowledge Unit (See NC-06 and NC-07)			
CYB/Web Security Knowledge Unit			
CYB-25. Explain browser and web security model concepts, including same-origin policy, web sessions, and secure communication channels. *[Understanding]*	Identify one or more browser and web security model concepts. *[Remembering]*	Explain browser and web security model concepts, including same-origin policy, web sessions, and secure communication channels, such as TLS. *[Understanding]*	Apply browser and web security model concepts including same-origin policy, web sessions, and secure communication channels, such as TLS. *[Applying]*

Discrete Structures Knowledge Area (DS)

The Discrete Structures (DS) knowledge area (KA) consists of 34 measurable student learning outcomes across six knowledge units. As indicated in Figure 6, most of the student learning outcomes, 50%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (35%) and *Analyzing* (15%) levels. The ACM CCECC estimates 40 professor/student contact hours for the CYB knowledge area.

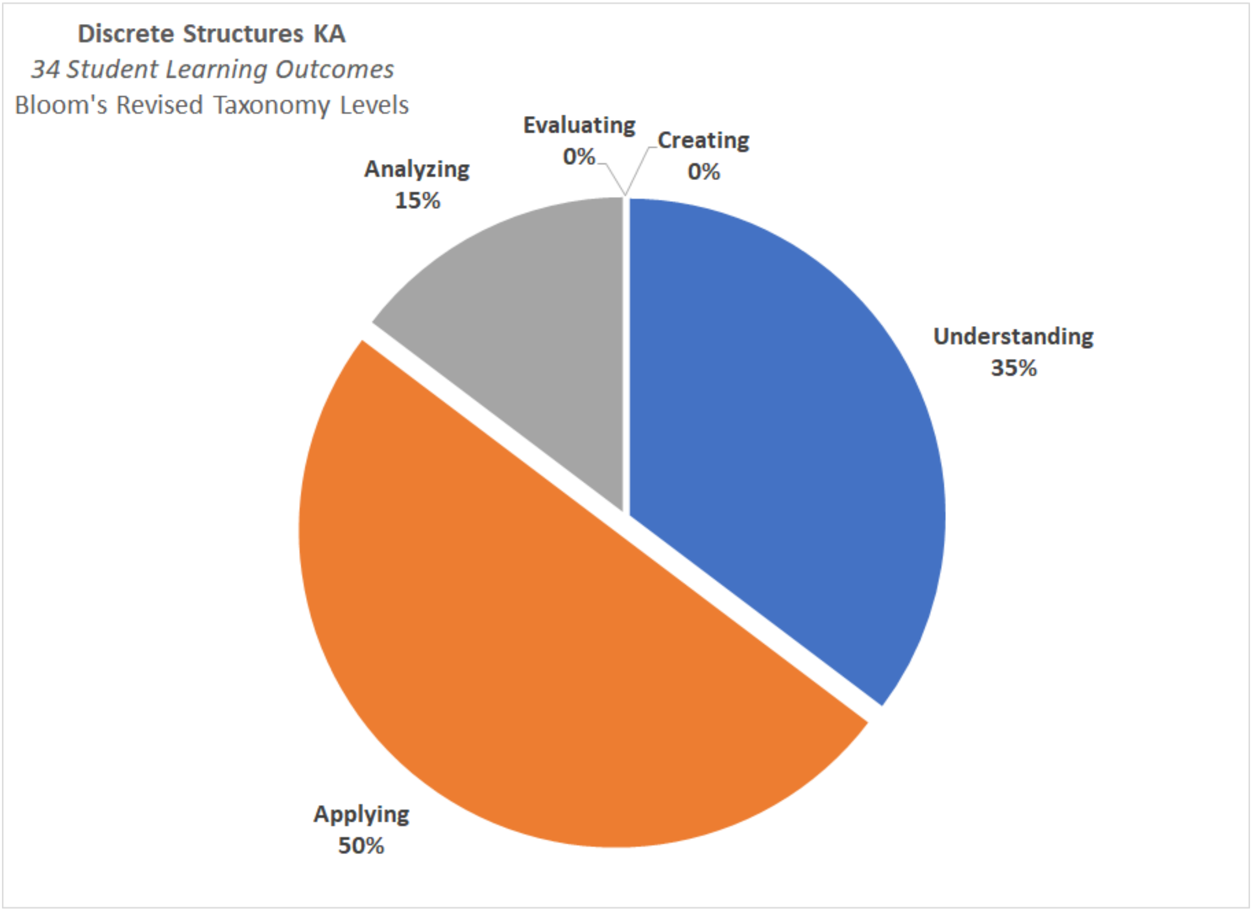

Figure 6 Pie Chart of Bloom's Revised Taxonomy Levels: DS KA LOs

Discrete structures serve as a foundation for many areas in computer science. Discrete structures include the study of logic, set theory, graph theory, and probability theory. The concepts studied in discrete structures are pervasive throughout the areas of data structures and algorithms, but also appear elsewhere in computer science. For example, an ability to create and understand a proof—either a formal symbolic proof or a less formal but still mathematically rigorous argument—is important in virtually every area of computer science, including formal specification, verification, databases, and cryptography. Graph theory concepts are used in networks, operating systems, and compilers. Set theory concepts are used in software engineering and in databases. Probability theory is used in intelligent systems, networking, and several computing applications. (*adapted from CS2013, p. 76*).

Learning Outcome	**Assessment Rubric**		
DS. Discrete Structures KA	**Emerging**	**Developed**	**Highly Developed**
DS/Sets, Relations, and Functions Knowledge Unit			
DS-01. Explain with examples the basic terminology of functions, relations, and sets. *[Understanding]*	Identify the defining features of functions, relations, and sets. *[Remembering]*	Explain with examples the basic terminology of functions, relations, and sets. *[Understanding]*	Use function, relations, and set terminology in a correct and meaningful way. *[Applying]*
DS-02. Perform the operations associated with sets, functions, and relations. *[Applying]*	Describe the operations associated with sets, functions, and relations. *[Understanding]*	Perform the operations associated with sets, functions, and relations. *[Applying]*	Compare the operations of sets, functions, and relations. *[Analyzing]*
DS-03. Compare practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context. *[Analyzing]*	Implement a solution to a programming problem using a specific set, function, or relation model. *[Applying]*	Compare practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context. *[Analyzing]*	Justify the choice of a specific set, function, or relation model. *[Evaluating]*
DS/Basic Logic Knowledge Unit			
DS-04. Convert logical statements from informal language to propositional and predicate logic expressions. *[Understanding]*	Recognize the relationship between logical statements from informal language and propositional and predicate logic expressions. *[Remembering]*	Convert logical statements from informal language to propositional and predicate logic expressions. *[Understanding]*	Produce propositional and predicate logic expressions from a given logical statement from an informal language. *[Applying]*
DS-05. 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]*	Describe the steps in formal logic proofs and/or informal logical reasoning to solve real problems. *[Understanding]*	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]*	Compare different logic proofs and informal logical reasoning to determine correct methods to solve real problems. *[Analyzing]*

DS-06. Use the rules of inference to construct proofs in propositional and predicate logic. *[Applying]*	Discuss the rules of inference to construct proofs in propositional and predicate logic. *[Understanding]*	Use the rules of inference to construct proofs in propositional and predicate logic. *[Applying]*	Analyze the rules of inference to construct proofs in propositional and predicate logic. *[Analyzing]*
DS-07. Describe how symbolic logic can be used to model real-life situations or computer applications. *[Understanding]*	List ways that symbolic logic can be used to model real-life situations or computer applications. *[Remembering]*	Describe how symbolic logic can be used to model real-life situations or computer applications, such as software analysis, database queries, and algorithms. *[Understanding]*	Use symbolic logic to model real-life situations or computer applications. *[Applying]*
DS-08. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms. *[Applying]*	Demonstrate formal methods of symbolic propositional and predicate logic. *[Understanding]*	Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms. *[Applying]*	Distinguish between formal methods of propositional and predicate logic to determine the most effective solutions to a given problem. *[Analyzing]*
DS-09. Describe the strengths and limitations of propositional and predicate logic. *[Understanding]*	List the strengths and limitations of propositional and predicate logic. *[Remembering]*	Describe the strengths and limitations of propositional and predicate logic. *[Understanding]*	Illustrate the strengths and limitations of propositional and predicate logic. *[Applying]*
DS/Proof Techniques Knowledge Unit			
DS-10. Outline the basic structure of each proof technique, including direct proof, proof by contradiction, and induction. *[Analyzing]*	Use the basic structure of each proof technique to solve a problem. *[Applying]*	Outline the basic structure of each proof technique, including direct proof, proof by contradiction, and induction. *[Analyzing]*	Choose the most effective proof technique to solve a problem. *[Evaluating]*
DS-11. Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument. *[Applying]*	Demonstrate each of the proof techniques by correctly constructing a sound argument. *[Understanding]*	Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument. *[Applying]*	Use each of the proof techniques correctly in the construction of a sound argument. *[Applying]*

DS-12. Deduce the best type of proof for a given problem. *[Analyzing]*	Compare the different proof methods. *[Analyzing]*	Deduce the best type of proof for a given problem. *[Analyzing]*	Construct a correct proof using the best method for a given problem. *[Creating]*
DS-13. Explain the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures. *[Understanding]*	Identify the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures. *[Remembering]*	Explain the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures. *[Understanding]*	Illustrate the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures. *[Applying]*
DS-14. Explain the relationship between weak and strong induction and give examples of the appropriate use of each. *[Understanding]*	Identify the relationship between weak and strong induction. *[Remembering]*	Explain the relationship between weak and strong induction and give examples of the appropriate use of each. *[Understanding]*	Solve problems using both weak and strong induction. *[Applying]*
DS/Basics of Counting Knowledge Unit			
DS-15. Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions. *[Applying]*	Describe counting arguments. *[Understanding]*	Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/ geometric progressions. *[Applying]*	Outline counting arguments. *[Analyzing]*
DS-16. Apply the pigeonhole principle in the context of a formal proof. *[Applying]*	Demonstrate the pigeonhole principle. *[Understanding]*	Apply the pigeonhole principle in the context of a formal proof. *[Applying]*	Analyze the pigeonhole principle in the context of a formal proof. *[Analyzing]*
DS-17. Calculate permutations and combinations of a set, and interpret the meaning in the context of the particular application. *[Applying]*	Explain the calculation of permutations and combinations of a set. *[Understanding]*	Calculate permutations and combinations of a set, and interpret the meaning in the context of the particular application. *[Applying]*	Discriminate between computation of sets, permutations and combinations. *[Analyzing]*

DS-18. Compare real-world applications appropriate to counting formalisms. *[Analyzing]*	Use counting formalisms to solve real-world applications. *[Applying]*	Compare real-world applications appropriate to counting formalisms, such as determining the number of ways to arrange people around a table, subject to constraints on the seating arrangement. *[Analyzing]*	Choose appropriate counting formalisms to solve real-world applications. *[Evaluating]*
DS-19. Solve a variety of basic recurrence relations. *[Applying]*	Demonstrate a variety of basic recurrence relations. *[Understanding]*	Solve a variety of basic recurrence relations. *[Applying]*	Compare a variety of basic recurrence relations. *[Analyzing]*
DS-20. Analyze a problem to determine underlying recurrence relations. *[Analyzing]*	Carry out a problem with an underlying recurrence relation. *[Applying]*	Analyze a problem to determine underlying recurrence relations. *[Analyzing]*	Evaluate a problem with an underlying recurrence relation. *[Evaluating]*
DS-21. Perform computations involving modular arithmetic. *[Applying]*	Discuss computations involving modular arithmetic. *[Understanding]*	Perform computations involving modular arithmetic. *[Applying]*	Examine computations involving modular arithmetic. *[Analyzing]*
DS/Graphs and Trees Knowledge Unit			
DS-22. Illustrate the basic terminology of graph theory including properties and special cases for each type of graph/tree. *[Applying]*	Describe the basic terminology of graph theory. *[Understanding]*	Illustrate the basic terminology of graph theory including properties and special cases for each type of graph/tree. *[Applying]*	Outline the basic terminology of graph theory. *[Analyzing]*
DS-23. Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. *[Understanding]*	List the different traversal methods for trees and graphs. *[Remembering]*	Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. *[Understanding]*	Execute different traversal methods for trees and graphs. *[Applying]*

DS-24. 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]*	Discuss a variety of real-world problems in computer science using appropriate forms of graphs and trees. *[Understanding]*	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]*	Distinguish between real-world problems solvable by using graphs and trees. *[Analyzing]*
DS-25. Implement and use balanced trees and B-trees. *[Applying]*	Explain balanced trees and B-trees. *[Understanding]*	Implement and use balanced trees and B-trees. *[Applying]*	Analyze the use of balanced trees and B-trees. *[Analyzing]*
DS-26. Implement graph algorithms. *[Applying]*	Classify graph algorithms. *[Understanding]*	Implement graph algorithms, such as graph search, spanning trees, and shortest paths. *[Applying]*	Categorize different implementations of graph algorithms. *[Analyzing]*
DS-27. Demonstrate how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting. *[Understanding]*	Identify how concepts from graphs and trees appear in data structures, algorithms, proof techniques, and counting. *[Remembering]*	Demonstrate how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting. *[Understanding]*	Implement data structures, algorithms, proof techniques, and counting using graphs and trees. *[Applying]*
DS-28. Describe binary search trees and AVL trees. *[Understanding]*	Define binary search and AVL trees. *[Remembering]*	Describe binary search trees and AVL trees. *[Understanding]*	Apply binary search and AVL trees. *[Applying]*
DS-29. Explain complexity in the ideal and in the worst-case scenario for both implementations. *[Understanding]*	State complexity in the ideal and in the worst-case scenario for both implementations. *[Remembering]*	Explain complexity in the ideal and in the worst-case scenario for both implementations. *[Understanding]*	Calculate complexity in the ideal and in the worst-case scenario for both implementations. *[Applying]*

DS/Discrete Probability Knowledge Unit			
DS-30. Calculate probabilities of events and expectations of random variables for elementary problems. *[Applying]*	Exemplify probabilities of events and expectations of random variables for elementary problems such as games of chance. *[Understanding]*	Calculate probabilities of events and expectations of random variables for elementary problems, such as games of chance. *[Applying]*	Examine probabilities of events and expectations of random variables for elementary problems such as games of chance. *[Analyzing]*
DS-31. Differentiate between dependent and independent events. *[Understanding]*	Identify dependent and independent events. *[Remembering]*	Differentiate between dependent and independent events. *[Understanding]*	Illustrate dependent and independent events. *[Applying]*
DS-32. Explain the significance of binomial distribution in probabilities. *[Understanding]*	Recognize the notation and parameters that a binomial distribution has. *[Remembering]*	Explain the significance of binomial distribution in probabilities. *[Understanding]*	Calculate the probabilities from a binomial distribution. *[Applying]*
DS-33. Apply Bayes Theorem to determine conditional probabilities in a problem. *[Applying]*	Explain Bayes Theorem to determine conditional probabilities in a problem. *[Understanding]*	Apply Bayes Theorem to determine conditional probabilities in a problem. *[Applying]*	Outline Bayes Theorem to determine conditional probabilities in a problem. *[Analyzing]*
DS-34. Apply the tools of probability to solve problems. *[Applying]*	Discuss the tools of probability to solve problems, such as the average case analysis of algorithms *[Understanding]*	Apply the tools of probability to solve problems, such as the average case analysis of algorithms. *[Applying]*	Analyze the tools of probability in solving problems such as the average case analysis of algorithms. *[Analyzing]*

Graphics and Visualization Knowledge Area (GV)

The Graphics and Visualization (GV) knowledge area (KA) consists of five measurable student learning outcomes in one knowledge unit. As indicated in Figure 7, most of the student learning outcomes, 40% and 40%, are at both the *Applying* and *Analyzing* levels of Bloom's Revised Taxonomy. The remaining learning outcomes are at the *Understanding* (20%) level. The ACM CCECC estimates two professor/student contact hours for the GV knowledge area.

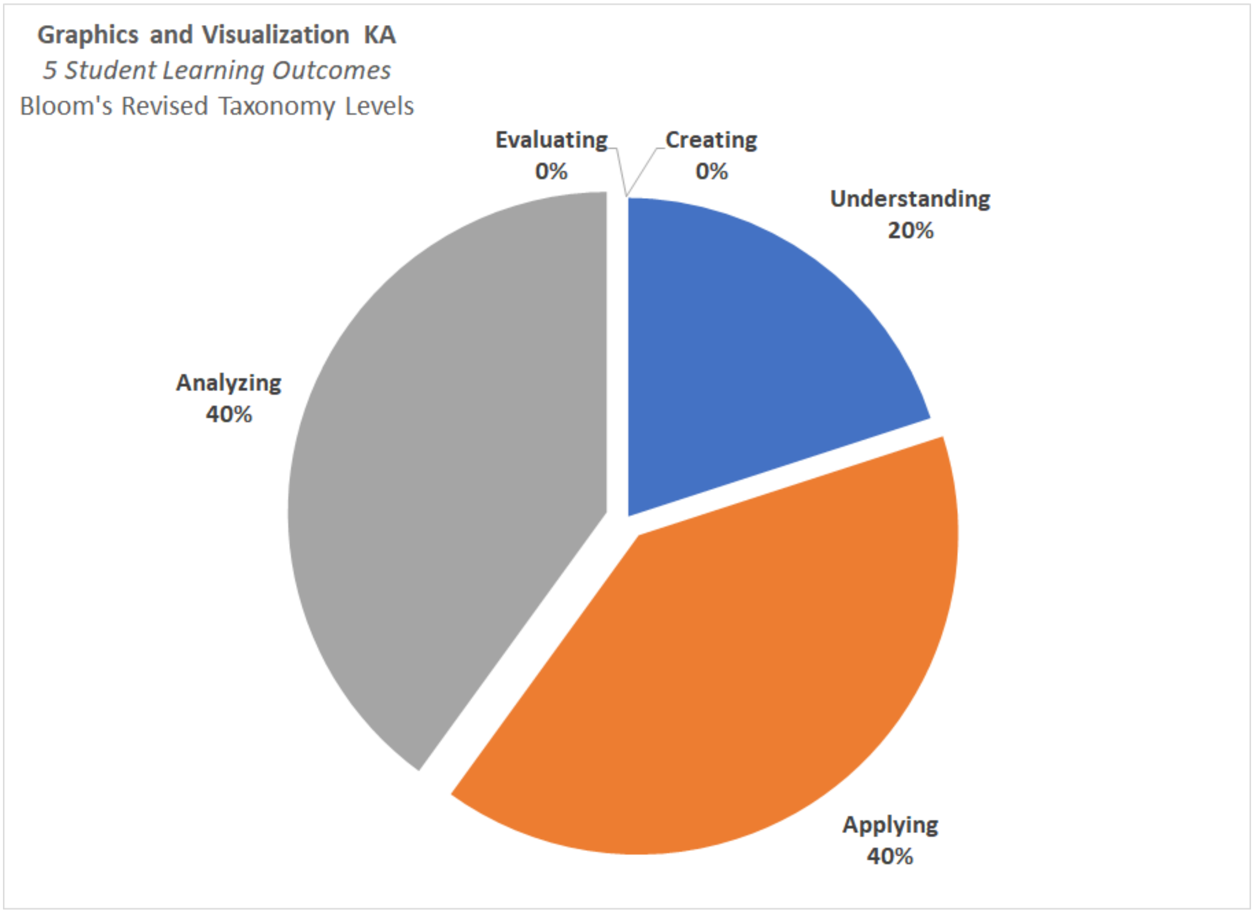

Figure 7 Pie Chart of Bloom's Revised Taxonomy Levels: GV KA LOs

Computer graphics is the term commonly used to describe the computer generation and manipulation of images. It is the science of enabling visual communication through computation, including cartoons, film special effects, videogames, medical imaging, engineering, as well as scientific, information, and knowledge visualization. Traditionally, graphics at the undergraduate level has focused on rendering, linear algebra, and phenomenological approaches. More recently, the focus has begun to include physics, numerical integration, scalability, and special-purpose hardware. In order for students to become adept at the use and generation of computer graphics, many implementation-specific issues must be addressed, such as file formats, hardware interfaces, and application program interfaces. (*adapted from CS2013, p. 82*).

Learning Outcome	**Assessment Rubric**		
GV. Graphics and Visualization KA	**Emerging**	**Developed**	**Highly Developed**
GV/Fundamental Concepts Knowledge Unit			
GV-01. Compare transformation and changes in dimension and coordinate systems for 2D and 3D design. *[Analyzing]*	Explain how to use dimensions and coordinate systems. *[Understanding]*	Compare transformation and changes in dimension and coordinate systems for 2D and 3D design. *[Analyzing]*	Evaluate the uses of both 3D coordinate systems and 2D planar systems with respect to computer graphics. *[Evaluating]*
GV-02. Demonstrate common uses of digital presentation to human senses. *[Understanding]*	Identify common uses of digital presentation to human senses. *[Remembering]*	Demonstrate common uses of digital presentation to human senses, such as computer graphics, sound, and haptic devices. *[Understanding]*	Categorize types of digital presentation to human senses. *[Analyzing]*
GV-03. Illustrate color models and their use in computer graphics. *[Applying]*	Describe color models and their use in computer graphics. *[Understanding]*	Illustrate color models and their use in computer graphics. *[Applying]*	Contrast color models and their use in computer graphics. *[Analyzing]*
GV-04. Analyze image types according to output choices. *[Analyzing]*	Differentiate multimedia file types, resolution needs, conversion, and appropriate use. *[Understanding]*	Analyze image types according to output choices. *[Analyzing]*	Evaluate multiple multimedia files based on given criteria. *[Evaluating]*
GV-05. Perform information hiding through steganography in images, messages, videos, or other media files. *[Applying]*	Demonstrate information hiding through steganography. *[Understanding]*	Perform information hiding through steganography in images, messages, videos, or other media files. *[Applying]*	Choose appropriate steganography technique to conceal information. *[Evaluating]*

Human-Computer Interaction Knowledge Area (HCI)

The Human-Computer Interaction (HCI) knowledge area (KA) consists of six measurable student learning outcomes across two knowledge units. As indicated in Figure 8, most of the student learning outcomes, 50% and 50%, are at both the *Applying* and *Analyzing* levels of Bloom's Revised Taxonomy. The ACM CCECC estimates five professor/student contact hours for the HCI knowledge area.

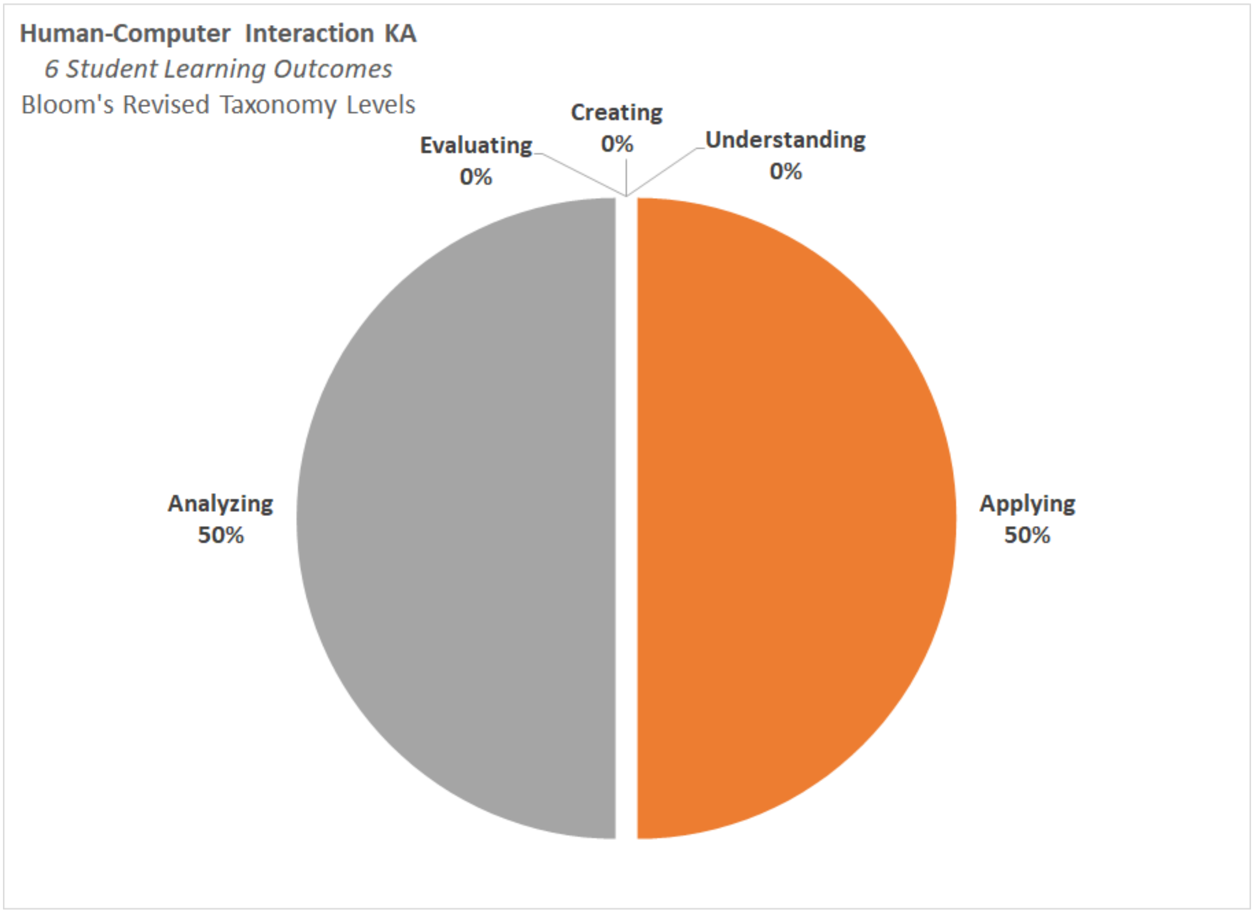

Figure 8 Pie Chart of Bloom's Revised Taxonomy Levels: HCI KA LOs

Human-computer interaction (HCI) studies cognitive science and human factors engineering as applied to computer science. HCI is concerned with designing interactions between human activities and the computational systems that support them, and with constructing interfaces to afford those interactions. Interaction between users and computational artifacts occurs at an interface that includes both software and hardware. Thus interface design impacts the software lifecycle in that it should occur early; the design and implementation of core functionality can influence the user interface – for better or worse. Because it deals with people as well as computational systems, the HCI knowledge area considers cultural, social, organizational, cognitive, and perceptual issues. Consequently, it draws on a variety of disciplinary traditions, including psychology, ergonomics, graphic and product design, anthropology and engineering. (*adapted from CS2013, p. 89*).

Learning Outcome	Assessment Rubric		
HCI. Human Computer Interaction KA	**Emerging**	**Developed**	**Highly Developed**
HCI/Foundations Knowledge Unit			
HCI-01. Analyze the importance of human-centered software. *[Analyzing]*	Discuss the importance of human-centered software development. *[Understanding]*	Analyze the importance of human-centered software. *[Analyzing]*	Defend the importance of human-centered software. *[Evaluating]*
HCI-02. Implement a simple usability test for an existing software application. *[Applying]*	Summarize usability testing. *[Understanding]*	Implement a simple usability test for an existing software application. *[Applying]*	Design a usability test for an existing software application. *[Creating]*
HCI-03. Examine the issues of trust in HCI, including examples of both high and low trust systems. *[Analyzing]*	Demonstrate design elements that make a human-computer interface trustworthy. *[Understanding]*	Examine the issues of trust in HCI, including examples of both high and low trust systems. *[Analyzing]*	Critique interface designs between high trust and low trust. *[Evaluating]*
HCI/Designing Interaction Knowledge Unit			
HCI-04. Write a simple application that uses a modern graphical user interface. *[Applying]*	Summarize the components of a modern graphical user interface. *[Understanding]*	Write a simple application that uses a modern graphical user interface. *[Applying]*	Examine a complete application that uses a modern graphical interface and documentation. *[Analyzing]*
HCI-05. Use at least one national or international user interface design standard in a simple application. *[Applying]* **See also SP-02.**	Identify national and international user interface design standards. *[Remembering]*	Use at least one national or international user interface design standard in a simple application, such as U.S. ADA Standards. *[Applying]*	Compare among national and international user interface design standards. *[Analyzing]*
HCI-06. Analyze the interaction between a security mechanism and its usability. *[Analyzing]*	Discuss potential usability issues related to a security mechanism. *[Understanding]*	Analyze the interaction between a security mechanism and its usability. *[Analyzing]*	Design a user interface that balances the tradeoffs between usability and security. *[Creating]*

Information Management Knowledge Area (IM)

The Information Management (IM) knowledge area (KA) consists of 13 measurable student learning outcomes across three knowledge units. As indicated in Figure 9, most of the student learning outcomes, 46%, are at the *Understanding* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Applying* (31%), *Analyzing* (15%), and *Creating* (8%) levels. The ACM CCECC estimates six professor/student contact hours for the IM knowledge area.

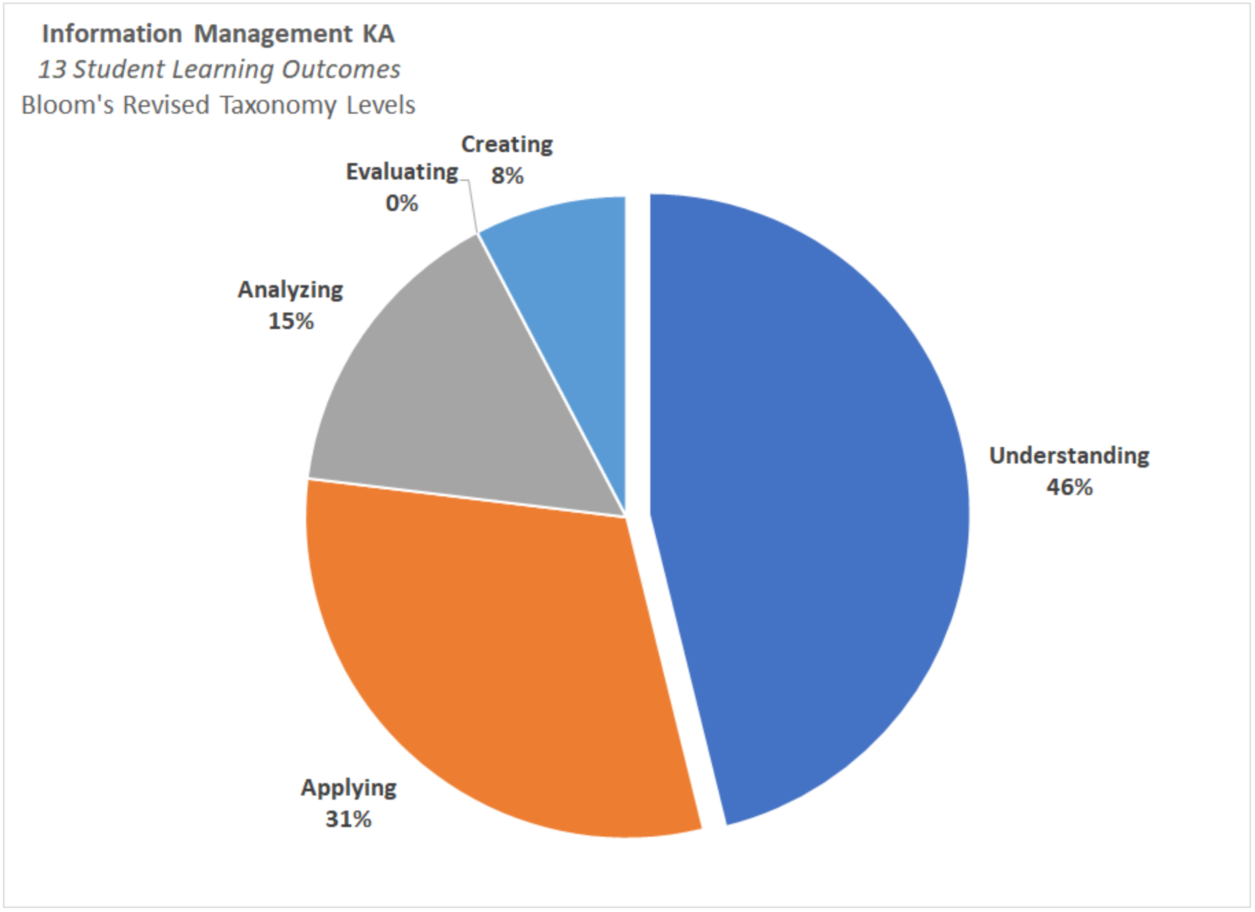

Figure 9 Pie Chart of Bloom's Revised Taxonomy Levels: IM KA LOs

Information Management is primarily concerned with the capture, digitization, representation, organization, transformation, and presentation of information; algorithms for efficient and effective access and updating of stored information; data modeling and abstraction; and physical file storage techniques. Students need the ability to develop conceptual and physical data models; determine which methods and techniques are appropriate for a given problem; and select and implement an appropriate solution that addresses relevant design concerns, including accessibility and usability. (*adapted from CS2013, p. 112).*

Learning Outcome	Assessment Rubric		
IM. Information Management KA	**Emerging**	**Developed**	**Highly Developed**
IM/Information Management Concepts Knowledge Unit			
IM-01. Contrast the difference between information and data. *[Analyzing]*	Explain information and data. *[Understanding]*	Contrast the difference between information and data. *[Analyzing]*	Critique the attributes of both information and data. *[Evaluating]*
IM-02. Describe the advantages and disadvantages of central organizational control over data. *[Understanding]*	List some advantages and disadvantages of central organizational control over data. *[Remembering]*	Describe the advantages and disadvantages of central organizational control over data. *[Understanding]*	Examine the advantages and disadvantages of central organizational control over data. *[Analyzing]*
IM-03. Investigate contingency planning with respect to business continuity, disaster recovery and incident response. *[Applying]*	Describe contingency planning with respect to business continuity, disaster recovery and incident response. *[Understanding]*	Investigate contingency planning with respect to business continuity, disaster recovery and incident response. *[Applying]*	Compare contingency plans of various size organizations with respect to business continuity, disaster recovery and incident response. *[Analyzing]*
IM-04. Describe proven techniques to secure data and information. *[Understanding]*	List proven techniques used to secure data and information. *[Remembering]*	Describe proven techniques to secure data and information. *[Understanding]*	Implement specific proven techniques to secure data and information. *[Applying]*
IM-05. Describe approaches to scale up information systems. *[Understanding]*	List approaches to scale up information systems. *[Remembering]*	Describe approaches to scale up information systems. *[Understanding]*	Compare several approaches to scale up information systems. *[Analyzing]*
IM/Database Systems Knowledge Unit			
IM-06. Explain the characteristics that distinguish the database approach from the approach of programming with data files. *[Understanding]*	Identify the characteristics that distinguish the database approach from the approach of programming with data files. *[Remembering]*	Explain the characteristics that distinguish the database approach from the approach of programming with data files. *[Understanding]*	Contrast the characteristics that distinguish the database approach from the approach of programming with data files. *[Analyzing]*

IM-07. Diagram the components of a database system and give examples of their use. *[Applying]*	Identify the components of a database system. *[Remembering]*	Diagram the components of a database system and give examples of their use. *[Applying]*	Organize the components of a database system into a secure, functioning system. *[Analyzing]*
IM-08. Explain the concept of data independence and its importance in a database system. *[Understanding]*	Define the concept of data independence and its importance in a database system. *[Remembering]*	Explain the concept of data independence and its importance in a database system. *[Understanding]*	Examine the concept of data independence and its importance in a database system. *[Analyzing]*
IM-09. Formulate queries in SQL or a similar query language to elicit information from a database. *[Creating]*	Edit queries in SQL or a similar query language to elicit information from a database. *[Applying]*	Formulate queries in SQL or a similar query language to elicit information from a database. *[Creating]*	Formulate complex queries in SQL or a similar query language to elicit information from a database. *[Creating]*
IM-10. Investigate vulnerabilities and failure scenarios in database systems. *[Applying]*	Identify vulnerabilities and failure scenarios in information systems. *[Remembering]*	Investigate vulnerabilities and failure scenarios in database systems, such as SQL injection and cross-site scripting. *[Applying]*	Examine vulnerabilities and failure scenarios in information systems. *[Analyzing]*
IM/Data Modeling Knowledge Unit			
IM-11. Contrast appropriate data models, including internal structures for different data types. *[Analyzing]*	Diagram appropriate data models, including internal structures for different data types. *[Applying]*	Contrast appropriate data models, including internal structures for different data types. *[Analyzing]*	Evaluate appropriate data models, including internal structures for different data types. *[Evaluating]*
IM-12. Diagram a relational data model for a given scenario that addresses security and privacy concerns. *[Applying]*	Describe a relational data model for a given scenario that addresses security and privacy concerns. *[Understanding]*	Diagram a relational data model for a given scenario that addresses security and privacy concerns. *[Applying]*	Analyze a relational data model for a given scenario that addresses security and privacy concerns. *[Analyzing]*
IM-13. Describe the differences among relational data models and other models. *[Understanding]*	Recognize the differences among relational data models and other models. *[Remembering]*	Describe the differences among relational data models and other models such as Object-Oriented, JSON, NoSQL. *[Understanding]*	Analyze the differences among relational data models and other models. *[Analyzing]*

Networking and Communications Knowledge Area (NC)

The Networking and Communications (NC) knowledge area (KA) consists of eight measurable student learning outcomes across two knowledge units. As indicated in Figure 10, most of the student learning outcomes, 50%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (37%) and *Analyzing* (13%) levels. The ACM CCECC estimates five professor/student contact hours for the NC knowledge area.

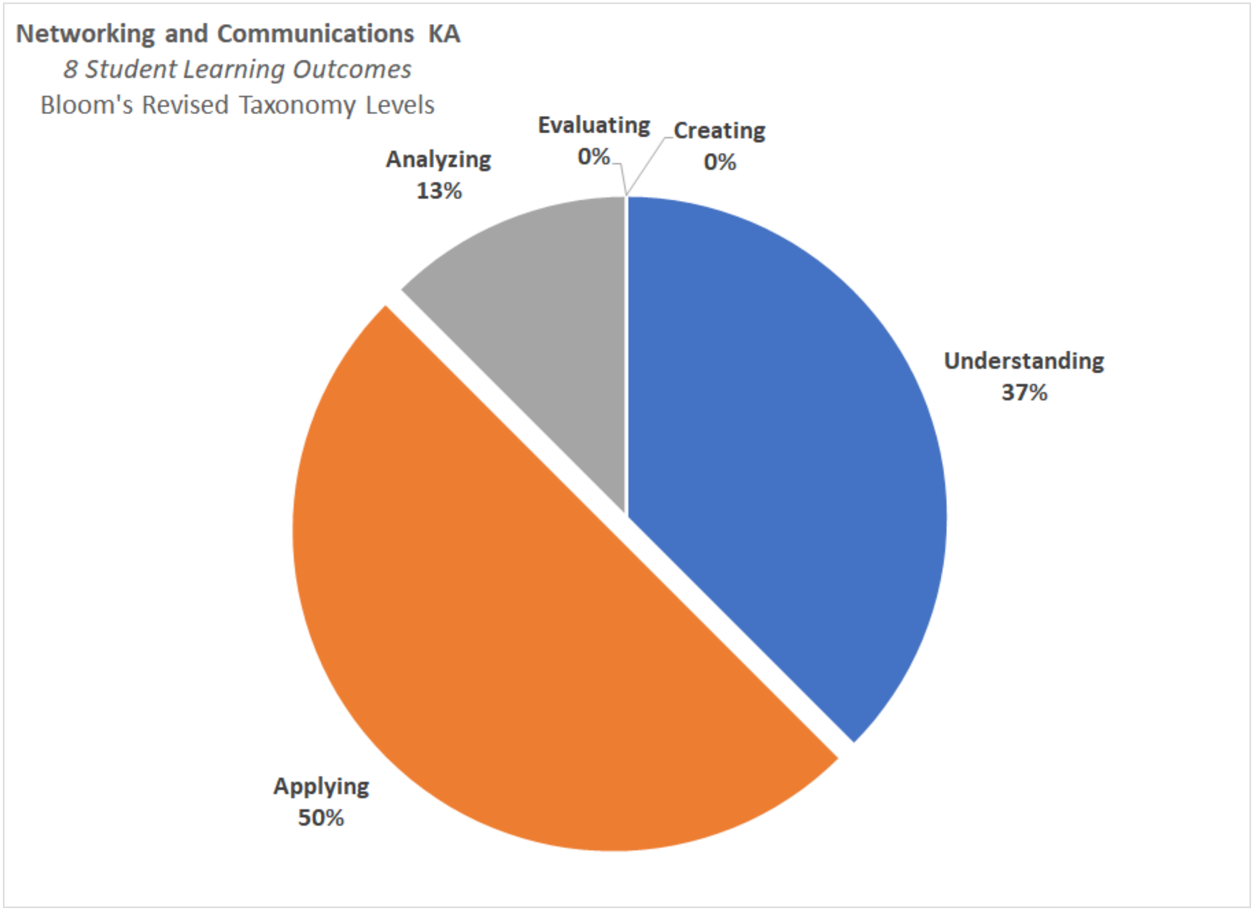

Figure 10 Pie Chart of Bloom's Revised Taxonomy Levels: NC KA LOs

The Internet, computer networks, cloud computing, and the Internet of Things (IoT) are now ubiquitous, and a growing number of computing activities strongly depend on the correct operation of an underlying network. Networks, both fixed and mobile, are a key part of the computing environment of today and tomorrow. Many computing applications that are used today would not be possible without networks. This dependency on an underlying network is likely to increase in the future. (*adapted from CS2013, p. 130*).

Learning Outcome	**Assessment Rubric**		
NC. Networking and Communications KA	**Emerging**	**Developed**	**Highly Developed**
NC/Introduction Knowledge Unit			
NC-01. Diagram the basic structure of the Internet. *[Applying]*	Explain the basic structure of the Internet. *[Understanding]*	Diagram the basic structure of the Internet. *[Applying]*	Analyze the basic structure of the internet. *[Analyzing]*
NC-02. Describe the layered structure of a typical networked architecture, including routing and switching. *[Understanding]*	Label the components of a typical networked architecture. *[Remembering]*	Describe the layered structure of a typical networked architecture, including routing and switching. *[Understanding]*	Diagram the layered structure of a typical networked architecture. *[Applying]*
NC-03. Diagram the layers of the OSI model, including associated protocols. *[Applying]*	Describe the layers of the OSI model. *[Understanding]*	Diagram the layers of the OSI model, including associated protocols. *[Applying]*	Compare the layers of the OSI model with the TCP/IP model. *[Analyzing]*
NC/Networked Applications Knowledge Unit			
NC-04. Categorize the principles used for naming schemes and resource location. *[Analyzing]*	Summarize principles used for naming schemes and resource location. *[Understanding]*	Categorize the principles used for naming schemes and resource location. *[Analyzing]*	Evaluate the principles used for naming schemes and resource location. *[Evaluating]*
NC-05. Implement a simple distributed network application. *[Applying]*	Identify the components of a simple distributed network application. *[Remembering]*	Implement a simple distributed network application. *[Applying]*	Integrate a simple distributed network application within a server program to exchange data with a client. *[Analyzing]*
NC-06. Describe security concerns in designing applications for use over wireless networks. *[Understanding]*	Recognize security concerns in designing applications for use over wireless networks. *[Remembering]*	Describe security concerns in designing applications for use over wireless networks. *[Understanding]*	Illustrate security concerns in designing applications for use over wireless networks. *[Applying]*
NC-07. Illustrate secure connectivity among networked applications. *[Applying]*	Explain secure connectivity among networked applications. *[Understanding]*	Illustrate secure connectivity among networked applications, such as SSH, HTTPS, SFTP. *[Applying]*	Critique secure connectivity among networked applications. *[Evaluating]*

NC-08. Explain the advantages and disadvantages of using virtualized infrastructure in Cloud computing. *[Understanding]* **See also PD-05.**	List the advantages and disadvantages of using virtualized infrastructure in Cloud computing. *[Remembering]*	Explain the advantages and disadvantages of using virtualized infrastructure in Cloud computing. *[Understanding]*	Investigate the advantages and disadvantages of using virtualized infrastructure in Cloud computing. *[Applying]*

Operating Systems Knowledge Area (OS)

The Operating Systems (OS) knowledge area (KA) consists of 13 measurable student learning outcomes across six knowledge units. As indicated in Figure 11, most of the student learning outcomes, 77%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are at the *Analyzing* (23%) level. The ACM CCECC estimates 10 professor/student contact hours for the OS knowledge area.

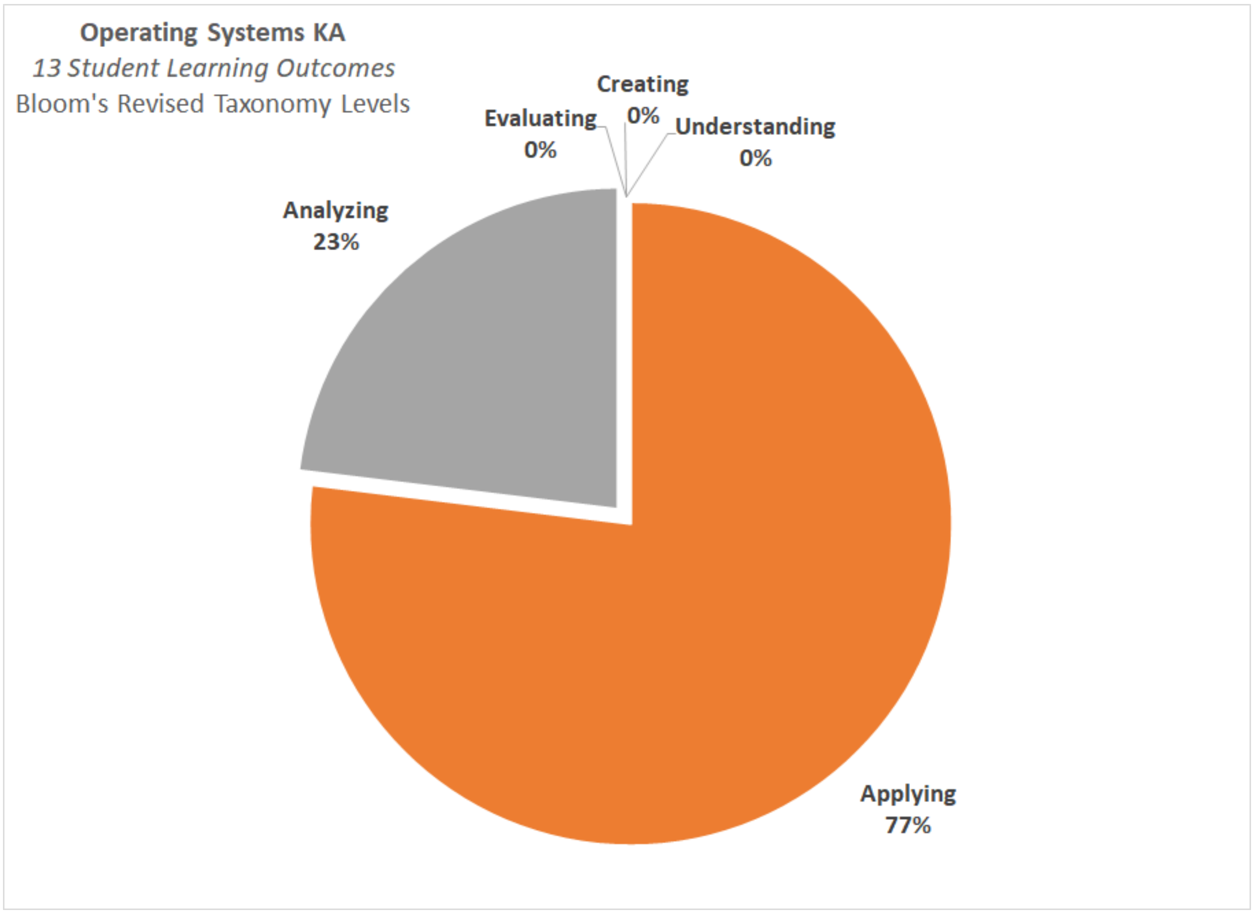

Figure 11 Pie Chart of Bloom's Revised Taxonomy Levels: OS KA LOs

An operating system defines an abstraction of hardware and manages resource sharing among the computer's users. The topics include the most basic knowledge of operating systems in the sense of interfacing an operating system to networks, teaching the difference between the kernel and user system design and implementation. This knowledge area is structured to be complementary to the Systems Fundamentals (SF), Networking and Communications (NC), and the Parallel and Distributed Computing (PD) knowledge areas. (*adapted from CS2013, p. 135*).

Learning Outcome	**Assessment Rubric**		
OS. Operating Systems KA	**Emerging**	**Developed**	**Highly Developed**
OS/Overview of Operating Systems Knowledge Unit			
OS-01. Examine major objectives, functions, features, and concepts of modern operating systems. *[Analyzing]*	Describe objectives, functions, features, and concepts of modern operating systems. *[Understanding]*	Examine major objectives, functions, features, and concepts of modern operating systems. *[Analyzing]*	Assess major objectives, functions, and concepts of modern operating systems. *[Evaluating]*
OS-02. Compare prevailing types of operating systems. *[Analyzing]*	Investigate prevailing types of operating systems. *[Applying]*	Compare prevailing types of operating systems, such as networked, mobile, embedded, and real-time. *[Analyzing]*	Assess prevailing types of operating systems. *[Evaluating]*
OS-03. Illustrate potential threats to operating systems and appropriate security measures. *[Applying]*	Describe potential threats to operating systems and appropriate security measures. *[Understanding]*	Illustrate potential threats to operating systems and appropriate security countermeasures. *[Applying]*	Examine potential threats to operating systems and appropriate security countermeasures. *[Analyzing]*
OS/Operating System Principles Knowledge Unit			
OS-04. Diagram the interaction of an Application Programming Interface (API) with an operating system. *[Applying]*	Summarize the interaction of an Application Programming Interface (API) with an operating system. *[Understanding]*	Diagram the interaction of an Application Programming Interface (API) with an operating system. *[Applying]*	Test the interaction of an Application Programming Interface (API) with an operating system. *[Evaluating]*
OS-05. Illustrate how computing resources are used by applications and managed by the operating system. *[Applying]*	Exemplify how computing resources are used by applications and managed by the operating system. *[Understanding]*	Illustrate how computing resources are used by applications and managed by the operating system. *[Applying]*	Test how computing resources are used by applications and managed by the operating system. *[Evaluating]*
OS-06. Manipulate a device list or driver I/O queue. *[Applying]*	Explain the purpose of a device list and driver I/O queue. *[Understanding]*	Manipulate a device list or driver I/O queue. *[Applying]*	Categorize device types of a modern operating system. *[Analyzing]*

OS/Concurrency Knowledge Unit			
OS-07. Investigate the need for concurrency within an operating system. *[Applying]* **See also PD-01.**	Describe the need for concurrency within an operating system. *[Understanding]*	Investigate the need for concurrency within an operating system. *[Applying]*	Analyze the need for concurrency within an operating system. *[Analyzing]*
OS/Memory Management Knowledge Unit			
OS-08. Illustrate the principles of memory management. *[Applying]*	Describe the principles of memory management. *[Understanding]*	Illustrate the principles of memory management, such as memory hierarchy and allocation, tradeoffs, and caching. *[Applying]*	Analyze the principles of memory management. *[Analyzing]*
OS-09. Illustrate the concepts of virtual memory, including paging, thrashing, and partitioning. *[Applying]*	Describe the concepts of virtual memory, including paging, thrashing, and partitioning. *[Understanding]*	Illustrate the concepts of virtual memory, including paging, thrashing, and partitioning. *[Applying]*	Examine the concepts of virtual memory, including paging, thrashing, and partitioning. *[Analyzing]*
OS/Security and Protection Knowledge Unit			
OS-10. Investigate the features and limitations of an operating system used to provide protection and security. *[Applying]*	Explain the features and limitation of an operating system used to provide protection and security. *[Understanding]*	Investigate the features and limitation of an operating system used to provide protection and security. *[Applying]*	Examine the features and limitation of an operating system used to provide protection and security. *[Analyzing]*
OS-11. Use mechanisms available in an operating system to control access to resources. *[Applying]*	Summarize mechanisms available in an operating system to control access to resources. *[Understanding]*	Use mechanisms available in an operating system to control access to resources. *[Applying]*	Test mechanisms available in an operating system to control access to resources. *[Evaluating]*
OS/Virtual Machines Knowledge Unit			
OS-12. Analyze the concept of virtualization with respect to hardware and software. *[Analyzing]*	Investigate the concept of virtualization with respect to hardware and software. *[Applying]*	Analyze the concept of virtualization with respect to hardware and software. *[Analyzing]*	Assess a given implementation of virtualization with respect to hardware and software. *[Evaluating]*

OS-13. Diagram the physical hardware devices and the virtual devices maintained by an operating system. *[Applying]*	Explain the relationship between the physical hardware devices and virtual devices maintained by an operating system. *[Understanding]*	Diagram the physical hardware devices and the virtual devices maintained by an operating system. *[Applying]*	Distinguish between the physical hardware devices and virtual devices used by an operating system for a given implementation of virtualization. *[Analyzing]*

Parallel and Distributed Computing Knowledge Area (PD)

The Parallel and Distributed Computing (PD) knowledge area (KA) consists of five measurable student learning outcomes across three knowledge units. As indicated in Figure 12, most of the student learning outcomes, 60%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are at the *Analyzing* (40%) level. The ACM CCECC estimates three professor/student contact hours for the PD KA.

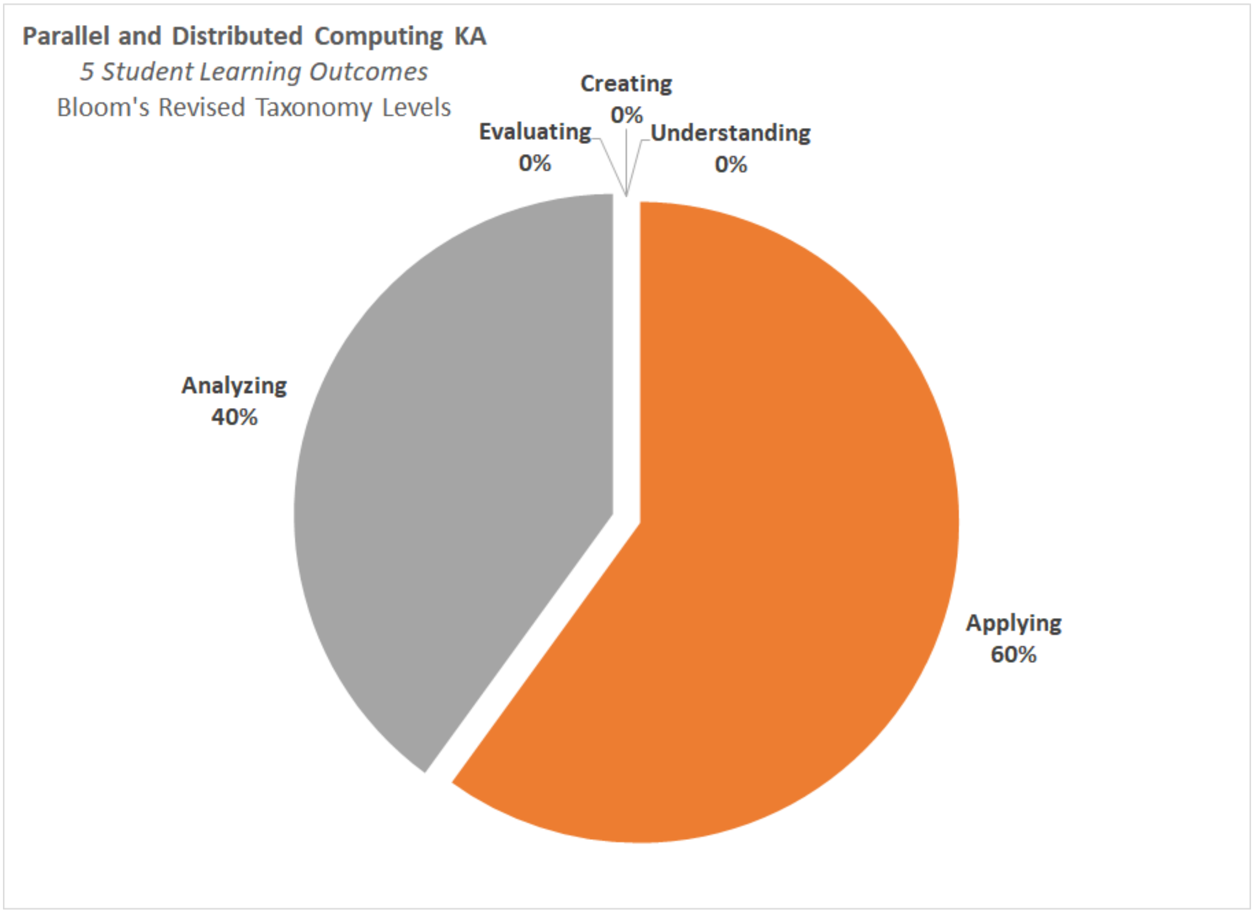

Figure 12 Pie Chart of Bloom's Revised Taxonomy Levels: PD KA LOs

The past decade has brought explosive growth in multiprocessor computing, including multi-core processors and distributed data centers. Both parallel and distributed computing entail the logically simultaneous execution of multiple processes, whose operations have the potential to interleave in complex ways. Parallel and distributed computing builds on foundations in many areas, including an understanding of fundamental systems concepts such as concurrency and parallel execution, consistency in state/memory manipulation, and latency. Communication and coordination among processes is rooted in the message-passing and shared-memory models of computing and such algorithmic concepts as atomicity, consensus, and conditional waiting. Distributed systems highlight the problems of security and fault tolerance, emphasize the maintenance of replicated state, and introduce additional issues that bridge computer networking. The Systems Fundamentals knowledge area also contains an introduction to parallelism (SF/Parallelism KU). The Operating Systems knowledge area also contains learning outcomes on concurrency (OS/Concurrency KU). (*adapted from CS2013, p. 145*).

Learning Outcome	**Assessment Rubric**		
PD. Parallel and Distributed Computing KA	**Emerging**	**Developed**	**Highly Developed**
PD/Parallelism Fundamentals Knowledge Unit			
PD-01. Analyze the goals of parallelism and concurrency. *[Analyzing]* **See also OS-07 and SF-08.**	Differentiate the goal of parallelism, such as throughput, from the goal of concurrency, such as controlling access to shared resources. *[Understanding]*	Analyze the goals of parallelism and concurrency. *[Analyzing]*	Evaluate the performance of a given program that was implemented using parallelism and concurrency techniques. *[Evaluating]*
PD-02. Implement various programming constructs for synchronization. *[Applying]*	Summarize various programming constructs for synchronization. *[Understanding]*	Implement various programming constructs for synchronization. *[Applying]*	Integrate synchronization routines/techniques into a non-synchronized programming constructs. *[Analyzing]*
PD-03. Contrast low-level data races with higher level races. *[Analyzing]*	Differentiate low-level data races from higher level races. *[Understanding]*	Contrast low-level data races with higher level races. *[Analyzing]*	Create a low-level data race among two concurrent threads. *[Creating]*
PD/Communication and Coordination Knowledge Unit			
PD-04. Implement mutual exclusion in order to avoid race conditions that could cause security vulnerabilities. *[Applying]*	Explain mutual exclusion in order to avoid race conditions. *[Understanding]*	Implement mutual exclusion in order to avoid race conditions that could cause security vulnerabilities. *[Applying]*	Categorize critical and noncritical race conditions. *[Analyzing]*

PD/Cloud Computing Knowledge Unit			
PD-05. Investigate the challenges and concerns related to security and privacy in Cloud computing. *[Applying]* **See also NC-08.**	Describe the challenges and concerns related to security and privacy in Cloud computing. *[Understanding]*	Investigate the challenges and concerns related to security and privacy in Cloud computing. *[Applying]*	Examine the challenges and concerns related to security and privacy in Cloud computing. *[Analyzing]*

Platform-based Development Knowledge Area (PBD)

There are no hours associated with the Platform-based Development (PBD) knowledge area in CS2013, but rather this KA 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 considers platform-specific constraints. For instance, web programming, multimedia development, mobile app development, the Internet of Things (IoT), and robotics are examples of relevant platforms that provide specific services/APIs/hardware that constrain development.” (*adapted from CS2013.org, p. 144*).

Programming Languages Knowledge Area (PL)

The Programming Languages (PL) knowledge area (KA) consists of 10 measurable student learning outcomes across four knowledge units. As indicated in Figure 13, most of the student learning outcomes, 50%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (20%), *Analyzing* (10%), and *Creating* (20%) levels. The ACM CCECC estimates 15 professor/student contact hours for the PL knowledge area.

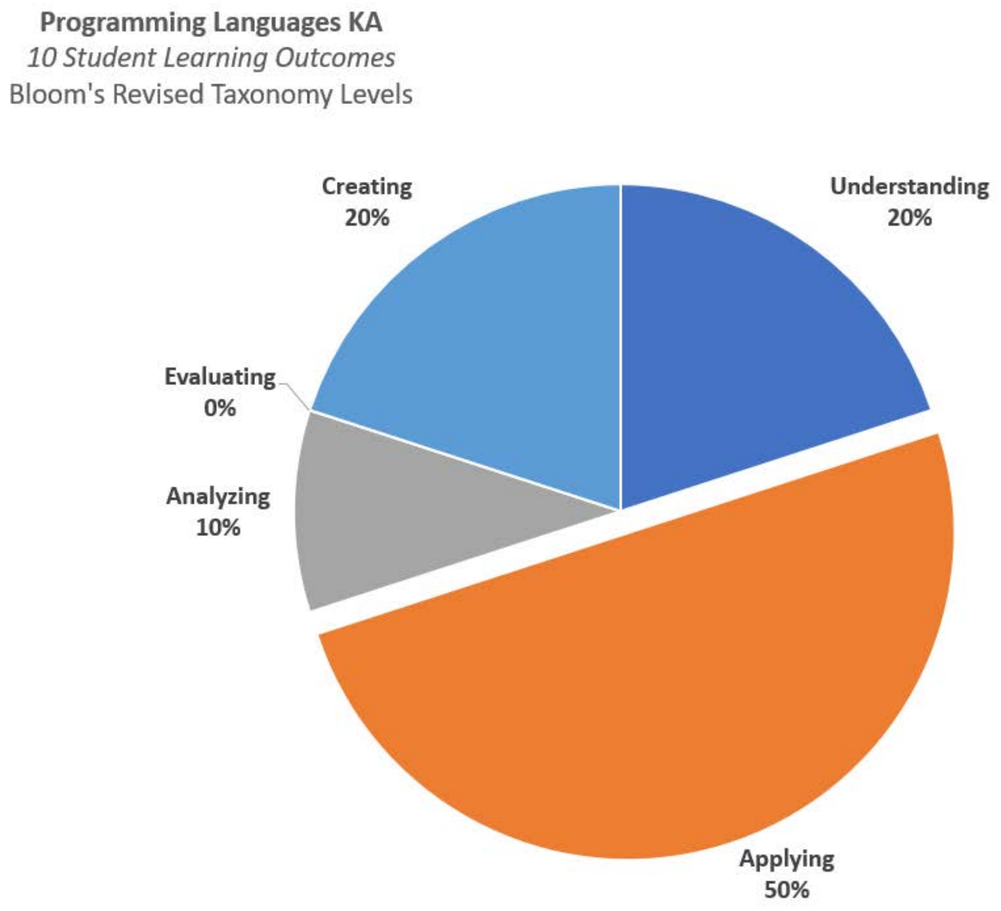

Figure 13 Pie Chart of Bloom's Revised Taxonomy Levels: PL KA LOs

Programming languages are the medium through which programmers precisely describe concepts, formulate algorithms, and reason about solutions. A computer scientist will work with many different languages, separately or together. Software developers must understand the programming models underlying different languages and make informed design choices in languages supporting multiple complementary approaches. Computer scientists will often need to learn new languages and programming constructs, and must understand the principles underlying how programming language features are defined, composed, and implemented. The effective use of programming languages, and appreciation of their limitations, also requires a basic knowledge of programming language translation and static program analysis, as well as run-time components and the memory management hierarchy. (*adapted from CS2013, p. 155*).

Learning Outcome	Assessment Rubric		
PL. Programming Languages KA	**Emerging**	**Developed**	**Highly Developed**
PL/Object-Oriented Programming Knowledge Unit			
PL-01. Design a simple class hierarchy using superclasses, subclasses, and abstract classes. *[Creating]*	Implement a simple class hierarchy using superclasses, subclasses, and abstract classes. *[Applying]*	Design a simple class hierarchy using superclasses, subclasses, and abstract classes. *[Creating]*	Develop a complex class hierarchy using superclasses, subclasses, and abstract classes. *[Creating]*
PL-02. Diagram control flow in a program using dynamic dispatch. *[Applying]*	Demonstrate control flow that uses dynamic dispatch. *[Understanding]*	Diagram control flow in a program using dynamic dispatch. *[Applying]*	Contrast control flow that uses a static environment vs a dynamic environment. *[Analyzing]*
PL-03. Use access and visibility modifiers to secure class data and methods. *[Applying]*	Describe access modifiers to secure class data such as private and protected. *[Understanding]*	Use access and visibility modifiers to secure class data and methods, such as private and protected. *[Applying]*	Analyze the security effect of using access and visibility modifiers in code. *[Analyzing]*
PL-04. Implement in code OOP constructs, including encapsulation, abstraction, inheritance, and polymorphism. *[Applying]*	Demonstrate the tenets of OOP, including encapsulation, abstraction, inheritance, and polymorphism. *[Understanding]*	Implement in code OOP constructs, including encapsulation, abstraction, inheritance, and polymorphism. *[Applying]*	Create a program that utilizes OOP constructs, including encapsulation, abstraction, inheritance, and polymorphism. *[Analyzing]*
PL/Functional Programming Knowledge Unit			
PL-05. Implement algorithms which utilize immutable and mutable variables. *[Applying]*	Discuss how functional languages handle both immutable and mutable variables. *[Understanding]*	Implement algorithms which utilize immutable and mutable variables. *[Applying]*	Evaluate the efficiency of different algorithms which utilize immutable vs mutable variables. *[Evaluating]*
PL-06. Contrast functional and object-oriented programming paradigms. *[Analyzing]*	Explain major differences between functional and object-oriented programming paradigms. *[Understanding]*	Contrast functional and object-oriented programming paradigms. *[Analyzing]*	Appraise functional and object-oriented programming paradigms. *[Evaluating]*

PL/Event-Driven and Reactive Programming Knowledge Unit			
PL-07. Create an interactive program using an event-driven style. *[Creating]*	Describe advantages of having an event-driven programming style vs a pre-defined programming style. *[Understanding]*	Create an interactive program using an event-driven style. *[Creating]*	Create a complex program using an event-driven style. *[Creating]*
PL-08. Describe potential security vulnerabilities in event-driven GUI applications. *[Understanding]*	Identify potential security vulnerabilities in event-driven GUI applications. *[Remembering]*	Describe potential security vulnerabilities in event-driven GUI applications, such as injection-based attacks. *[Understanding]*	Illustrate potential security vulnerabilities in event-driven GUI applications. *[Applying]*
PL/Basic Type Systems Knowledge Unit			
PL-09. Investigate potential errors detected from both strong-type and weak-type languages. *[Applying]*	Summarize possible errors detected from both strong-type and weak-type languages. *[Understanding]*	Investigate possible errors detected from both strong-type and weak-type languages. *[Applying]*	Discriminate among errors detected from strong-type and from weak-type languages. *[Analyzing]*
PL-10. Explain the security implications of a type-safe language for software development. *[Understanding]*	Recognize the security implications of a type-safe language for software development. *[Remembering]*	Explain the security implications of a type-safe language for software development. *[Understanding]*	Examine the security advantages and disadvantages of a type-safe language for software development. *[Analyzing]*

Software Development Fundamentals Knowledge Area (SDF)

The Software Development Fundamentals (SDF) knowledge area (KA) consists of 19 measurable student learning outcomes across four knowledge units. As indicated in Figure 14, most of the student learning outcomes, 47%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Analyzing* (16%) and *Creating* (37%) levels. The ACM CCECC estimates 30 professor/student contact hours for the SDF KA.

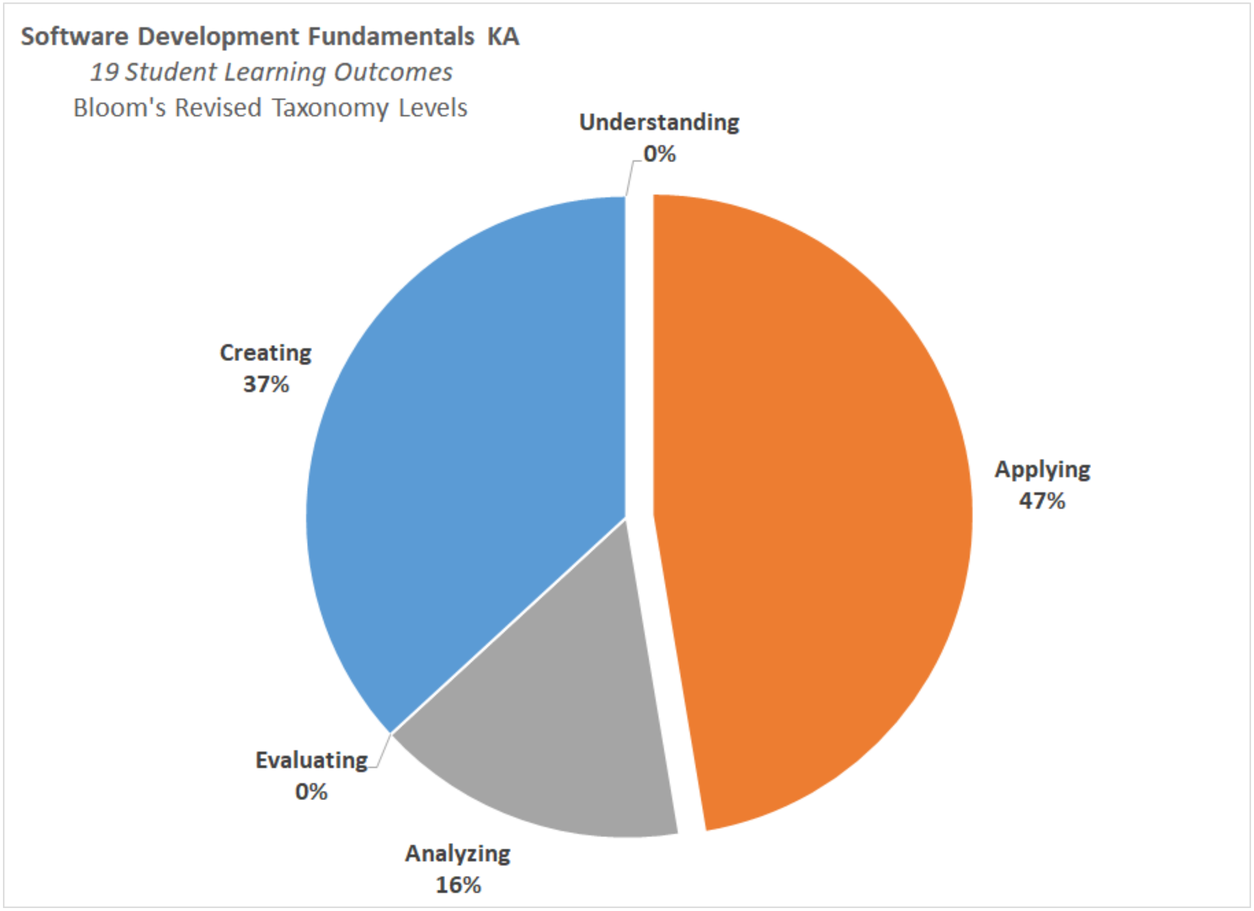

Figure 14 Pie Chart of Bloom's Revised Taxonomy Levels: SDF KA LOs

Fluency in the process of software development is a prerequisite to the study of most of computer science. To use computers to solve problems effectively, students must be competent at reading and writing programs in multiple programming languages. Beyond programming skills, however, they must be able to design and analyze algorithms, select appropriate paradigms, and utilize modern development and testing tools. This knowledge area brings together those fundamental concepts and skills related to the software development process. As such, it provides a foundation for other software-oriented knowledge areas, most notably Programming Languages, Algorithms and Complexity, and Software Engineering. (*adapted from CS2013, p. 167*).

Learning Outcome	**Assessment Rubric**		
SDF. Software Development Fundamentals KA	**Emerging**	**Developed**	**Highly Developed**
SDF/Algorithms and Design Knowledge Unit (see also AL/Algorithm Strategies and AL/Fundamental Data Structures and Algorithms)			
SDF-01. Design an algorithm in a programming language to solve a simple problem. *[Creating]*	Implement an algorithm in a programming language to solve a simple problem. *[Applying]*	Design an algorithm in a programming language to solve a simple problem. *[Creating]*	Design an algorithm in a programming language to solve a complex problem. *[Creating]*
SDF-02. Use the techniques of decomposition to modularize a program. *[Applying]*	Explain program decomposition. *[Understanding]*	Use the techniques of decomposition to modularize a program. *[Applying]*	Analyze code to see how decomposition techniques were used. *[Analyzing]*
SDF-03. Compare multiple algorithms for a given problem. *[Analyzing]*	Investigate multiple algorithms for a given problem. *[Applying]*	Compare multiple algorithms for a given problem. *[Analyzing]*	Evaluate the strengths and weaknesses of multiple algorithms for a problem. *[Evaluating]*
SDF/Fundamental Programming Concepts Knowledge Unit			
SDF-04. Create simple programs that use abstract data types (ADTs). *[Creating]*	Implement simple programs that use abstract data types (ADTs). *[Applying]*	Create simple programs that use abstract data types (ADTs). *[Creating]*	Write complex programs that use abstract data types (ADTs). *[Creating]*
SDF-05. Investigate potential vulnerabilities in provided programming code. *[Applying]* **See also AL-16, PL-10, SDF-12.**	Summarize potential vulnerabilities in programming code. *[Understanding]*	Investigate potential vulnerabilities in provided programming code. *[Applying]*	Choose a solution to mitigate vulnerabilities in programming code. *[Evaluating]*
SDF-06. Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. *[Creating]* **See also CYB-15.**	Investigate defensive programming techniques. *[Applying]*	Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. *[Creating]*	Create complex programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow. *[Creating]*

SDF-07. Create code in a programming language that includes primitive data types, references, variables, expressions, assignments, I/O, control structures, and functions. *[Creating]*	Implement code in a programming language that includes primitive data types, references, variables, expressions, assignments, I/O, control structures, and functions. *[Applying]*	Create code in a programming language that includes primitive data types, references, variables, expressions, assignments, I/O, control structures, and functions. *[Creating]*	Create complex programs that include primitive data types, references, variables, expressions, assignments, I/O, control structures, and functions. *[Creating]*
SDF-08. Create a simple program that uses persistence to save data across multiple executions. *[Creating]*	Implement a simple program that uses persistence to save data across multiple executions. *[Applying]*	Create a simple program that uses persistence to save data across multiple executions. *[Creating]*	Create a complex program that uses persistence to save data across multiple executions. *[Creating]*
SDF-09. Create a simple program that uses recursion. *[Creating]*	Implement a simple program that uses recursion. *[Applying]*	Create a simple program that uses recursion. *[Creating]*	Develop a complex program that includes various types of recursive techniques, such as binary, tail, and natural recursion. *[Creating]*
SDF/Fundamental Data Structures Knowledge Unit (see also AL/Fundamental Data Structures and Algorithms)			
SDF-10. Create simple programs that include each of the following data structures: lists, stacks, queues, hash tables, graphs and trees. *[Creating]*	Implement programs that include each of the following data structures: lists, stacks, queues, hash tables, graphs and trees. *[Applying]*	Create simple programs that include each of the following data structures: lists, stacks, queues, hash tables, graphs and trees. *[Creating]*	Create complex programs that include each of the following data structures: lists, stacks, queues, hash tables, graphs and trees. *[Creating]*
SDF-11. Compare the efficiency of basic operations across various data structures. *[Analyzing]*	Investigate the efficiency of basic operations across various data structures. *[Applying]*	Compare the efficiency of basic operations, such as insertion and deletion, across various data structures. *[Analyzing]*	Critique the efficiency of basic operations across various data structures. *[Evaluating]*

SDF/Development Methods Knowledge Unit			
SDF-12. Investigate common coding errors that introduce security vulnerabilities *[Applying]*	Describe common coding errors that expose security vulnerabilities. *[Understanding]*	Investigate common coding errors that introduce security vulnerabilities, such as buffer overflows, integer errors, and memory leaks. *[Applying]*	Test for common coding errors that introduce security vulnerabilities and the associated techniques for securing the code. *[Evaluating]*
SDF-13. Implement refactoring within given program components. *[Applying]*	Recognize refactoring opportunities within given program components. *[Remembering]*	Implement refactoring within given program components. *[Applying]*	Create program components utilizing refactoring. *[Applying]*
SDF-14. Analyze programming code that utilizes preconditions, postconditions, and invariants. *[Analyzing]*	Describe programming by contract. *[Understanding]*	Analyze programming code that utilizes preconditions, postconditions, and invariants. *[Analyzing]*	Create a program utilizing preconditions, postconditions, and invariants. *[Creating]*
SDF-15. Apply a variety of strategies to test and debug programs. *[Applying]*	Explain strategies to test and debug programs. *[Understanding]*	Apply a variety of strategies to test and debug programs, such as unit testing and test-case generation. *[Applying]*	Analyze a variety of strategies to test and debug programs. *[Analyzing]*
SDF-16. Use an integrated development environment (IDE) to create, execute, test, and debug secure programs. *[Applying]*	Discuss the benefits of using an integrated development environment (IDE) to create, execute, test, and debug secure programs. *[Understanding]*	Use an integrated development environment (IDE) to create, execute, test, and debug secure programs. *[Applying]*	Compare integrated development environments (IDEs) for a given programming language. *[Analyzing]*
SDF-17. Use standard libraries for a given programming language. *[Applying]*	Describe standard libraries for a given programming language. *[Understanding]*	Use standard libraries for a given programming language. *[Applying]*	Choose appropriate components from standard libraries to solve a given problem. *[Evaluating]*
SDF-18. Apply consistent documentation and program style standards. *[Applying]*	Explain the reasons for using consistent documentation and program style standards. *[Understanding]*	Apply consistent documentation and program style standards. *[Applying]*	Assess documentation and program style in a given program. *[Evaluating]*

SDF-19. Carry out a code review on a program component using a provided security checklist. *[Applying]*	Explain the process of a code review. *[Understanding]*	Carry out a code review on a program component using a provided security checklist. *[Applying]*	Organize a team code review on a program component using a provided security checklist. *[Analyzing]*

Software Engineering Knowledge Area (SE)

The Software Engineering (SE) knowledge area (KA) consists of 14 measurable student learning outcomes across seven knowledge units. As indicated in Figure 15, most of the student learning outcomes, 65%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the Understanding (14%), *Analyzing* (14%), and *Creating* (7%) levels. The ACM CCECC estimates 15 professor/student contact hours for the SE knowledge area.

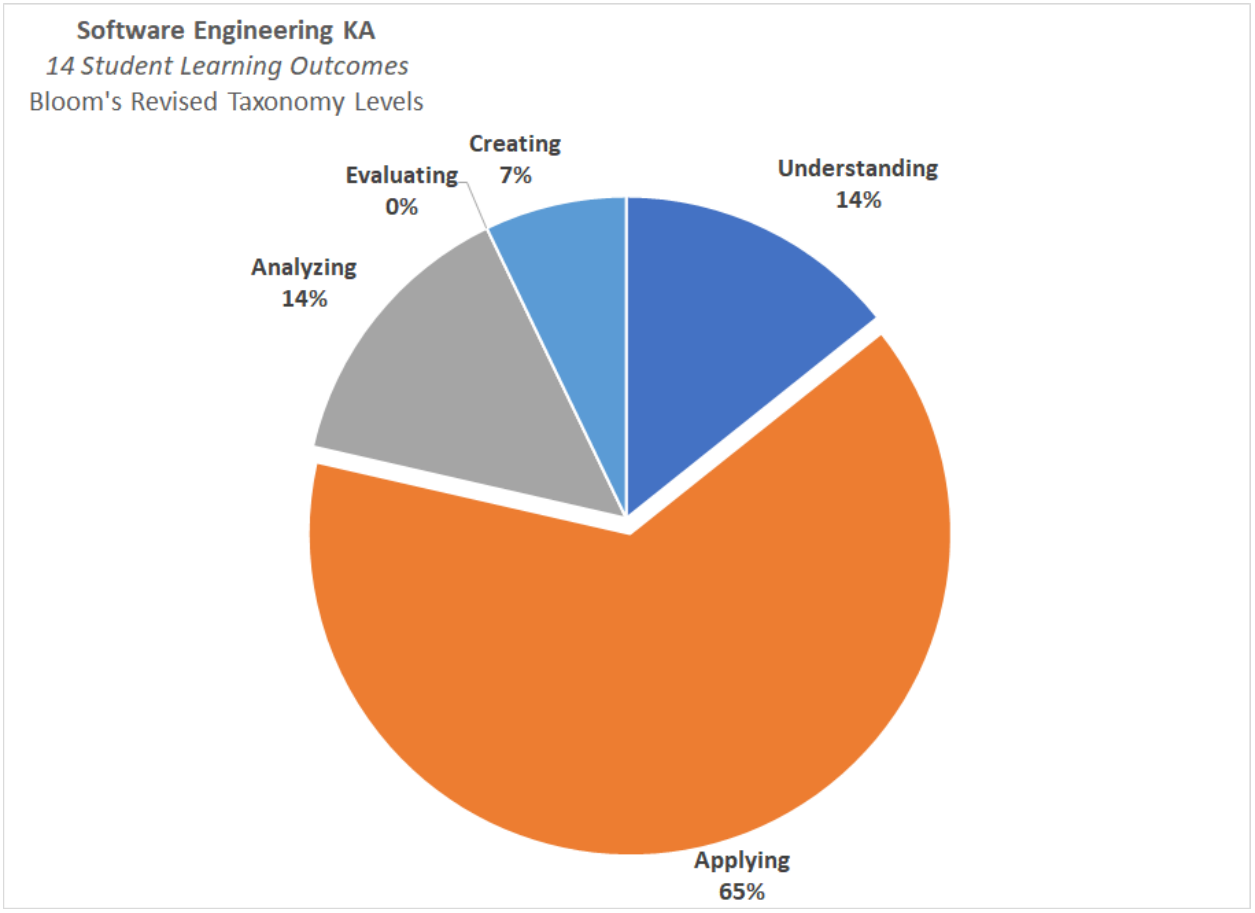

Figure 15 Pie Chart of Bloom's Revised Taxonomy Levels: SE KA LOs

Software engineering is the discipline concerned with the application of theory, knowledge, and practice to effectively and efficiently build reliable software systems that satisfy the requirements of customers and users. This discipline is applicable to small, medium, and large-scale systems. It encompasses all phases of the lifecycle of a software system, including requirements elicitation, analysis and specification; design; construction; verification and validation; deployment; and operation and maintenance. Whether small or large, following a traditional plan-driven development process, an agile approach, or some other method, software engineering is concerned with the best way to build good software systems.

Software engineering uses engineering methods, processes, techniques, and measurements. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. The software engineering toolbox has evolved over the years. Software development, which can involve an individual developer or a team or teams of developers, requires choosing the most appropriate tools, methods, and approaches for a given development environment. Practicing software engineers must select and apply appropriate techniques and practices to a given development effort in order to maximize value. (*adapted from CS2013, p. 172*).

Learning Outcome	**Assessment Rubric**		
SE. Software Engineering KA	**Emerging**	**Developed**	**Highly Developed**
SE/Software Processes Knowledge Unit			
SE-01. Diagram how software interacts with various systems, including information management, embedded, process control, and communications systems. *[Applying]*	Describe how software interacts with various systems including information management, embedded, process control, and communications systems. *[Understanding]*	Diagram how software interacts with various systems, including information management, embedded, process control, and communications systems. *[Applying]*	Assess how software interacts with various systems, including information management, embedded, process control, and communications systems. *[Evaluating]*
SE-02. Compare the features of various process models. *[Analyzing]*	Describe the features of a process model. *[Understanding]*	Compare the features of various process models, such as waterfall, iterative, and agile. *[Analyzing]*	Critique various process models. *[Evaluating]*
SE-03. Diagram the phases of the secure software development lifecycle (SecSDLC). *[Applying]*	Exemplify the phases the software development lifecycle (SecSDLC). *[Understanding]*	Diagram the phases of the secure software development lifecycle (SecSDLC). *[Applying]*	Examine the phases of the secure software development lifecycle (SecSDLC). *[Analyzing]*

SE/Software Project Management Knowledge Unit			
SE-04. Illustrate common behaviors that contribute to the effective functioning of a team. *[Applying]*	Describe common behaviors that contribute to the effective functioning of a team. *[Understanding]*	Illustrate common behaviors that contribute to the effective functioning of a team, such as good communication skills. *[Applying]*	Examine common behaviors that contribute to the effective functioning of a team. *[Analyzing]*
SE-05. Investigate the risks in using third-party applications, software tools, and libraries. *[Applying]*	Explain the risks in using third-party code. *[Understanding]*	Investigate the risks in using third-party applications, software tools, and libraries. *[Applying]*	Evaluate the risks in using third-party applications, software tools, and libraries. *[Evaluating]*
SE/Tools and Environments Knowledge Unit			
SE-06. Use a set of development tools for software systems. *[Applying]*	Summarize a set of development tools for software systems. *[Understanding]*	Use a set of development tools for software systems, such as requirements tracking, modeling, version control, automation, and testing. *[Applying]*	Choose a set of development tools for software systems. *[Evaluating]*
SE/Requirements Engineering Knowledge Unit			
SE-07. Implement the requirements for a secure software system. *[Applying]*	Paraphrase the requirements for a key feature for a secure software system. *[Understanding]*	Implement the requirements for a secure software system. *[Applying]*	Develop the requirements for a secure software system. *[Creating]*
SE/Software Design Knowledge Unit			
SE-08. Illustrate principles of secure software design. *[Applying]*	Describe different software design principles. *[Understanding]*	Illustrate principles of secure software design, such as least privilege, simplicity, separation of concerns, information hiding, coupling and cohesion, and code reuse. *[Applying]*	Create a program that employs secure software design principles. *[Creating]*

SE-09. Analyze an existing software design to improve its security. *[Analyzing]*	Identify possible stages of software design that may introduce a security vulnerability. *[Remembering]*	Analyze an existing software design to improve its security. *[Analyzing]*	Debate whether a proposed solution/ patch to the design can fix the vulnerability in a viable and effective way. *[Evaluating]*
SE-10. Describe the cost and tradeoffs associated with designing security into software. *[Understanding]*	Recognize situations where security designs are effectively applied in software. *[Remembering]*	Describe the cost and tradeoffs associated with designing security into software. *[Understanding]*	Compare security software designs and associated costs and tradeoffs. *[Analyzing]*
SE/Software Construction Knowledge Unit			
SE-11. Implement a small software project that uses a defined coding standard. *[Applying]*	Demonstrate a defined coding standard in a small software project. *[Understanding]*	Implement a small software project that uses a defined coding standard. *[Applying]*	Justify the reason for using a given coding standard. *[Evaluating]*
SE/Software Verification and Validation Knowledge Unit			
SE-12. Differentiate between program validation and verification. *[Understanding]*	Define software engineering terms verification and validation. *[Remembering]*	Differentiate between program validation and verification. *[Understanding]*	Apply software validation and verification for a given piece of code. *[Applying]*
SE-13. Implement in code different types of testing, including security, unit testing, system testing, integration testing, and interface usability. *[Applying]*	Describe different types and levels of testing. *[Understanding]*	Implement in code different types of testing, including security, unit testing, system testing, integration testing, and interface usability. *[Applying]*	Examine different types of testing for given code. *[Analyzing]*
SE-14. Design a test plan that validates software security. *[Creating]*	Implement a given test plan that validates software security. *[Applying]*	Design a test plan that validates software security. *[Creating]*	Develop a test plan that validates software security. *[Creating]*

Systems Fundamentals Knowledge Area (SF)

The Systems Fundamentals (SF) knowledge area (KA) consists of nine measurable student learning outcomes across three knowledge units. As indicated in Figure 16, most of the student learning outcomes, 67%, are at the *Applying* level of Bloom's Revised Taxonomy. The remaining learning outcomes are at the *Understanding* (33%) level. The ACM CCECC estimates five professor/student contact hours for the SF knowledge area.

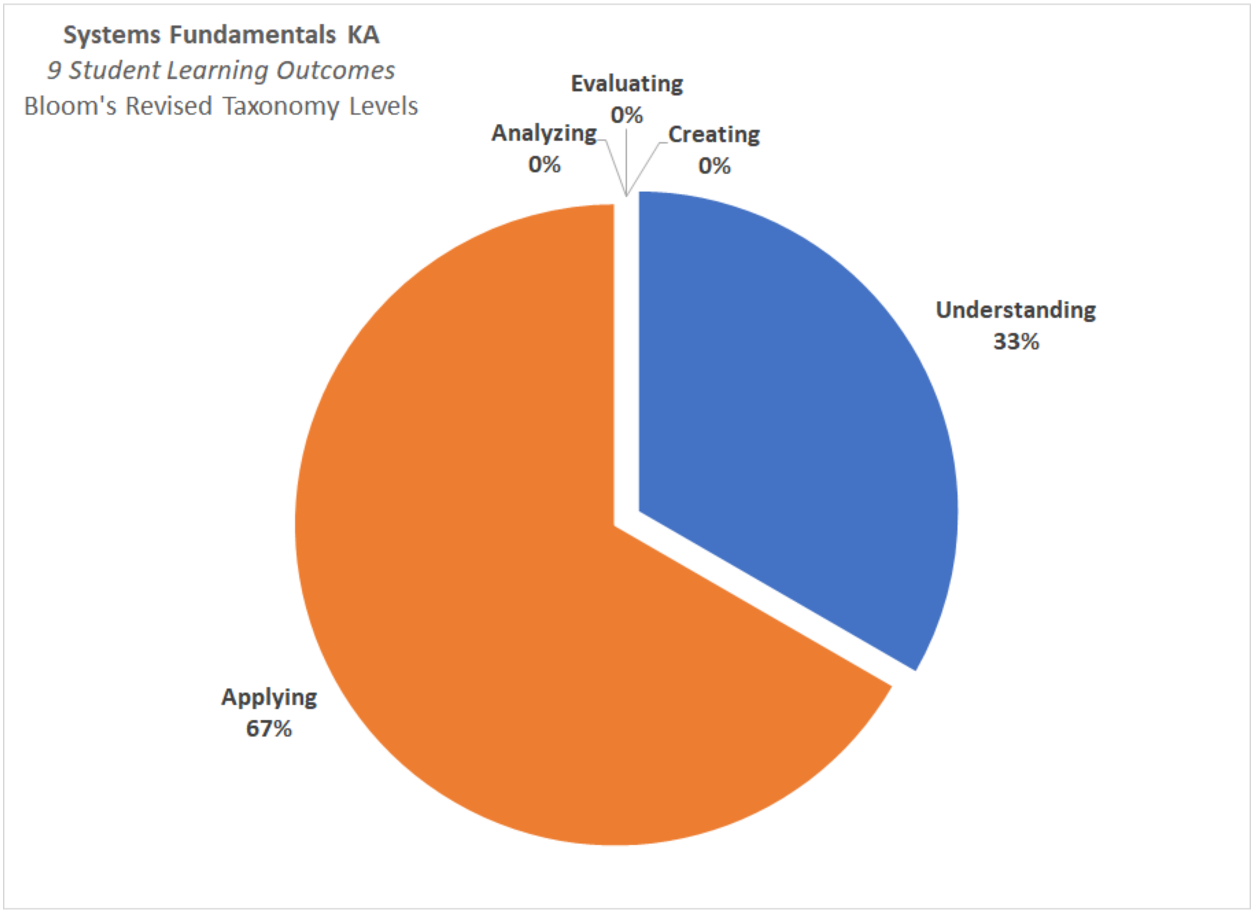

Figure 16 Pie Chart of Bloom's Revised Taxonomy Levels: SF KA LOs

The underlying hardware and software infrastructure upon which applications are constructed is collectively described by the term "computer systems." Computer systems broadly span the sub-disciplines of operating systems, parallel and distributed systems, communications networks, and computer architecture. Traditionally, these areas are taught in a non-integrated way through independent courses. However, these sub-disciplines increasingly share important common fundamental concepts within their respective cores. These concepts include computational paradigms, parallelism, cross-layer communications, state and state transition, resource allocation and scheduling, and so on. The Systems Fundamentals Knowledge Area is designed to present an integrative view of these fundamental concepts in a unified albeit simplified fashion, providing a common foundation for the different specialized mechanisms and policies appropriate to the specific domain area. (*adapted from CS2013, p. 186*).

Learning Outcome	Assessment Rubric		
SF. Systems Fundamentals KA	**Emerging**	**Developed**	**Highly Developed**
SF/Computational Paradigms Knowledge Unit			
SF-01. Illustrate the basic building blocks of computers and their role in the historical development of computer architecture. *[Applying]*	Describe some of the basic building blocks of computers. *[Understanding]*	Illustrate the basic building blocks of computers and their role in the historical development of computer architecture. *[Applying]*	Outline the basic building blocks of computers and their role in the historical development of computer architecture. *[Analyzing]*
SF-02. Discuss the differences between single thread and multiple thread, as well as single server and multiple server models. *[Understanding]*	Identify some differences between single thread and multiple thread, as well as single server and multiple server models. *[Remembering]*	Discuss the differences between single thread and multiple thread, as well as single server and multiple server models. *[Understanding]*	Illustrate the differences between single thread and multiple thread, as well as single server and multiple server models. *[Applying]*
SF-03. Investigate security implications related to emerging computational paradigms. *[Applying]*	Report security implications related to computational paradigms. *[Understanding]*	Investigate security implications related to emerging computational paradigms, such as quantum computing and biological computing. *[Applying]*	Analyze security implications related to emerging computational paradigms. *[Analyzing]*
SF/Cross-Layer Communications Knowledge Unit			
SF-04. Describe how computing systems are constructed of layers upon layers. *[Understanding]*	Recognize that computing systems are constructed of layers upon layers. *[Remembering]*	Describe how computing systems are constructed of layers upon layers, such as separation of concerns, well-defined interfaces, and abstraction. *[Understanding]*	Diagram a computing system constructed of layers upon layers. *[Applying]*
SF-05. Implement a program using methods of layering. *[Applying]*	Exemplify a program that uses methods of layering. *[Understanding]*	Implement a program using methods of layering, such as error detection, recovery and status across layers. *[Applying]*	Develop a program using methods of layering. *[Creating]*

SF-06. Investigate defects in a layered program using tools for program tracing, single stepping, and debugging. *[Applying]*	Demonstrate defects in a layered program using tools for program tracing, single stepping, and debugging. *[Understanding]*	Investigate defects in a layered program using tools for program tracing, single stepping, and debugging. *[Applying]*	Categorize defects by security risk in a layered program using tools for program tracing, single stepping, and debugging. *[Analyzing]*
SF/Parallelism Knowledge Unit			
SF-07. Illustrate the performance of simple sequential and parallel versions of the same program with different problem sizes. *[Applying]*	Summarize the general performance of simple sequential and parallel versions of the same program. *[Understanding]*	Illustrate the performance of simple and parallel versions of the same program with different problem sizes. *[Applying]*	Compare the performance of simple and parallel versions of the same program with different problem sizes. *[Analyzing]*
SF-08. Summarize the differences among the concepts of instruction parallelism, data parallelism, thread parallelism/multitasking, and task/request parallelism. *[Understanding]* **See also PD-01.**	Define the concepts of instruction parallelism, data parallelism, thread parallelism/multi-tasking, and task/request parallelism. *[Remembering]*	Summarize the differences among the concepts of instruction parallelism, data parallelism, thread parallelism/multitasking, and task/request parallelism. *[Understanding]*	Investigate the differences among the concepts of instruction parallelism, data parallelism, thread parallelism/ multitasking, and task/request parallelism. *[Applying]*
SF-09. Investigate other uses of parallelism, including reliability and redundancy of execution. *[Applying]*	Describe other uses of parallelism, including reliability and redundancy of execution. *[Understanding]*	Investigate other uses of parallelism, including reliability and redundancy of execution. *[Applying]*	Examine other uses of parallelism, including reliability and redundancy of execution. *[Analyzing]*

Social Issues and Professional Practice Knowledge Area (SP)

The Social Issues and Professional Practice (SP) knowledge area (KA) consists of 22 measurable student learning outcomes across eight knowledge units. As indicated in Figure 17, most of the student learning outcomes, 45%, are at the *Analyzing* level of Bloom's Revised Taxonomy. The remaining learning outcomes are shared among the *Understanding* (14%), *Applying*, (32%), *Evaluating* (4%), and *Creating* (5%) levels. The ACM CCECC estimates 12 professor/student contact hours for the SP knowledge area.

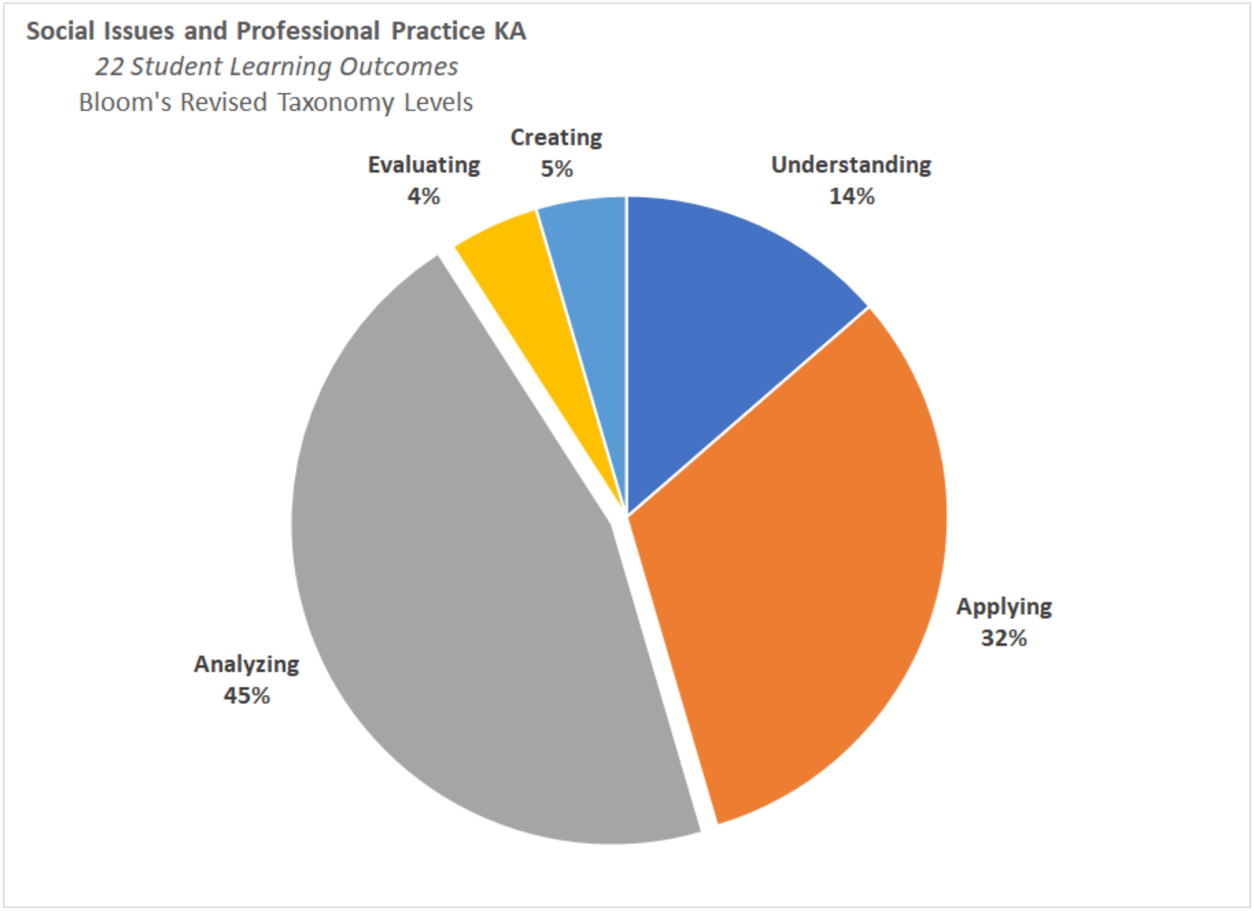

Figure 17 Pie Chart of Bloom's Revised Taxonomy Levels: SP KA LOs

As technological advances continue to significantly impact the way we live and work, the critical importance of social issues and professional practice continues to increase. While technical issues are central to the computing curriculum, they do not constitute a complete educational program in the field. Students must also be exposed to the larger societal context of computing to develop an understanding of the relevant social, ethical, legal and professional issues.

Students furthermore need to develop the ability to ask serious questions about the social impact of computing and to evaluate proposed answers to those questions. Future practitioners must be able to anticipate the impact of introducing a given product into a given environment. Will that product enhance or degrade the quality of life? What will the impact be upon individuals, groups, and institutions? Finally, students need to be aware of the basic legal rights of software and hardware vendors and users, and they also need to appreciate the ethical values that are the basis for those rights. Future practitioners must understand the responsibility that they will bear, and the possible consequences of failure. They must understand their own limitations as well as the limitations of their tools. All practitioners must make a long-term commitment to remaining current in their chosen specialties and in the complete discipline of computing.

The application of ethical analysis and reasoning underlies every subsection of the Social Issues and Professional Practices knowledge area in computing. The ACM Code of Ethics and Professional Conduct (ACM, 1992) provides guidelines that serve as the basis for the conduct of our professional work. The General Moral Imperatives provide an understanding of our commitment to personal responsibility, professional conduct, and our leadership roles. Computing faculty who are unfamiliar with the content and/or pedagogy of applied ethics are urged to take advantage of the considerable resources from ACM and its Special Interest Group on Computers and Society (SIGCAS). (*adapted from CS2013, p. 192*).

Learning Outcome	**Assessment Rubric**		
SP. Social Issues and Professional Practice KA	**Emerging**	**Developed**	**Highly Developed**
SP/Social Context Knowledge Unit			
SP-01. Investigate both positive and negative ways in which computing technology impacts information exchange and social interaction. *[Applying]*	Describe different ways in which computing technology impacts information exchange and social interaction. *[Understanding]*	Investigate both positive and negative ways in which computing technology impacts information exchange and social interaction, such as the Internet, mobile computing, and social media. *[Applying]*	Analyze positive and negative ways in which computing technology impacts information exchange and social interactions. *[Analyzing]*
SP-02. Examine developers' assumptions and values embedded in hardware and software design, especially with respect to underrepresented groups and diverse populations. *[Analyzing]* **See also HCI-06.**	Infer developers' assumptions and values embedded in hardware and software design, especially with respect to underrepresented groups and diverse populations. *[Understanding]*	Examine developers' assumptions and values embedded in hardware and software design, especially with respect to underrepresented groups and diverse populations, such as persons with disabilities. *[Analyzing]*	Critique developers' assumptions and values embedded in hardware and software design, especially as pertinent to underrepresented groups and the disabled. *[Evaluating]*
SP-03. Analyze the impact of diversity on the computing profession. *[Analyzing]*	Discuss the impact of diversity on the computing profession. *[Understanding]*	Analyze the impact of diversity on the computing profession, such as industry culture and product development. *[Analyzing]*	Assess the impact of diversity on the computing profession. *[Evaluating]*

SP-04. Investigate social engineering attacks and the types of bad actors who might perform them. *[Applying]*	Describe social engineering attacks and the types of bad actors who might perform them. *[Understanding]*	Investigate social engineering attacks and the types of bad actors who might perform them. *[Applying]*	Analyze the impact and likelihood of social engineering attacks. *[Analyzing]*
SP/Analytical Tools Knowledge Unit			
SP-05. Contrast stakeholder positions in a given scenario. *[Analyzing]*	Infer stakeholder positions in a given scenario. *[Understanding]*	Contrast stakeholder positions in a given scenario. *[Analyzing]*	Debate stakeholder positions in a given scenario. *[Evaluating]*
SP-06. Analyze social tradeoffs in technical decisions. *[Analyzing]*	Explain social tradeoffs in technical decisions. *[Understanding]*	Analyze ethical and social tradeoffs in technical decisions. *[Analyzing]*	Justify social tradeoffs in technical decisions. *[Evaluating]*
SP/Professional Ethics Knowledge Unit			
SP-07. Examine various ethics scenarios in computing. *[Analyzing]*	Discuss ethics scenarios in computing. *[Understanding]*	Analyze various ethics scenarios in computing. *[Analyzing]*	Debate various ethics scenarios in computing. *[Evaluating]*
SP-08. Support the ethical responsibility of ensuring software correctness, reliability, and safety. *[Evaluating]*	Examine the ethical responsibility in ensuring software correctness, reliability, and safety. *[Analyzing]*	Support the ethical responsibility of ensuring software correctness, reliability, and safety. *[Evaluating]*	Hypothesize various ethical responsibilities of ensuring software correctness, reliability, and safety. *[Creating]*
SP-09. Compare professional codes of conduct from the ACM, IEEE Computer Society, and other organizations. *[Analyzing]*	Discuss professional codes of conduct from the ACM, IEEE Computer Society, and other organizations. *[Understanding]*	Compare professional codes of conduct from the ACM, IEEE Computer Society, and other organizations. *[Analyzing]*	Evaluate professional codes of conduct from the ACM, IEEE Computer Society, and other organizations. *[Evaluating]*
SP/Intellectual Property Knowledge Unit			
SP-10. Differentiate among intellectual property, fair-use, copyright, patent, trademark, and plagiarism. *[Understanding]*	Define the terms intellectual property, fair-use, copyright, and plagiarism. Give examples of each. State the plagiarism policy at your school. *[Remembering]*	Differentiate among intellectual property, fair-use, copyright, patent, trademark, and plagiarism. *[Understanding]*	Investigate ethics violations related to intellectual property rights, fair-use, copyright, patents, trademarks, and plagiarism. *[Applying]*
SP-11. Discuss the rationale for legal protection of intellectual property.	Recognize the rationale for legal	Discuss the rationale for legal protection of intellectual property.	Examine the rationale for legal protection of intellectual property.

[Understanding]	protection of intellectual property. *[Remembering]*	*[Understanding]*	*[Analyzing]*
SP/Privacy and Civil Liberties Knowledge Unit			
SP-12. Outline the need for legal protection of personal privacy. *[Analyzing]*	Discuss the need for legal protection of personal privacy. *[Understanding]*	Outline the need for legal protection of personal privacy. *[Analyzing]*	Defend the need for legal protection of personal privacy. *[Evaluating]*
SP-13. Investigate threats to privacy rights in personally identifiable information (PII). *[Applying]*	Summarize threats to privacy rights in personally identifiable information (PII). *[Understanding]*	Investigate threats to privacy rights in personally identifiable information (PII). *[Applying]*	Analyze solutions for privacy threats to personally identifiable Information (PII). *[Analyzing]*
SP-14. Illustrate the role of data collection in the implementation of pervasive surveillance systems. *[Applying]*	Discuss the role of data collection in the implementation of pervasive surveillance systems. *[Understanding]*	Illustrate the role of data collection in the implementation of pervasive surveillance systems, such as RFID, face recognition, and mobile computing. *[Applying]*	Assess the role of data collection in the implementation of pervasive surveillance systems. *[Evaluating]*
SP-15. Analyze technological solutions to privacy concerns. *[Analyzing]*	Investigate technological solutions to privacy concerns. *[Applying]*	Analyze technological solutions to privacy concerns. *[Analyzing]*	Choose a technological solution to solve a privacy problem. *[Evaluating]*
SP/Professional Communication Knowledge Unit			
SP-16. Use effective oral, written, electronic, and visual communication techniques with stakeholders. *[Applying]*	Demonstrate effective oral, written, electronic, and visual communication techniques. *[Understanding]*	Use effective oral, written, electronic, and visual communication techniques with stakeholders. *[Applying]*	Choose the appropriate oral, written, electronic or visual communication technique with stakeholders. *[Evaluating]*
SP-17. Interpret the impact of both verbal and nonverbal cues during communication among team members. *[Understanding]*	Recognize both verbal and nonverbal cues during communication among team members. *[Remembering]*	Interpret the impact of both verbal and nonverbal cues during communication among team members. *[Understanding]*	Analyze the impact of both verbal and nonverbal cues during communication among team members. *[Analyzing]*
SP-18. Develop technical artifacts. *[Creating]*	Write a technical artifact. *[Applying]*	Develop technical artifacts, such as documentation of source code and user requirements, as well	Create technical artifacts of considerable length and/or complexity. *[Creating]*

		as project documents. *[Creating]*	
SP-19. Analyze case studies related to sustainable computing efforts. *[Analyzing]*	Paraphrase case studies related to sustainable computing efforts. *[Understanding]*	Analyze case studies related to sustainable computing efforts. *[Analyzing]*	Critique case studies related to sustainable computing efforts. *[Evaluating]*
SP/Security Policies, Laws and Computer Crime Knowledge Unit			
SP-20. Investigate laws applicable to computer crimes. *[Applying]*	Paraphrase laws applicable to computer crimes. *[Understanding]*	Investigate laws applicable to computer crimes. *[Applying]*	Debate laws applicable to computer crimes. *[Evaluating]*
SP-21. Examine the motivation and ramifications of cyber terrorism and criminal hacking. *[Analyzing]*	Discuss the motivation and ramifications of cyber terrorism and criminal hacking. *[Understanding]*	Examine the motivation and ramifications of cyber terrorism and criminal hacking. *[Analyzing]*	Evaluate the motivation and ramifications of cyber terrorism and criminal hacking. *[Evaluating]*
SP-22. Write a company-wide security policy. *[Applying]*	Exemplify a company-wide security policy. *[Understanding]*	Write a company-wide security policy, such as procedures for managing passwords, avoiding social engineering attacks, and monitoring employees. *[Applying]*	Compare several company-wide security policies. *[Analyzing]*

Table 5 Number of Student Learning Outcomes in each Knowledge Unit

Algorithms and Complexity KA		**Architecture and Organization KA**	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Basic Analysis	4	Digital Logic and Digital Systems	1
Algorithmic Strategies	2	Machine Level Representation of Data	4
Fundamental Data structures and Algorithms	8	Assembly Level Machine Organization	4
Basic Automata, Computability, and Complexity	3	Memory System Organization and Architecture	2

Computational Science KA		Cybersecurity KA	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Introduction to Modeling and Simulation	3	Foundational Concepts in Security	10
		Principles of Secure Design	4
		Defensive Programming	4
		Threats and Attacks	2
		Cryptography	4
		Web Security	1

Discrete Structures KA		Graphics and Visualization KA	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Sets, Relations, and Functions	3	Fundamental Concepts	5
Basic Logic	6		
Proof Techniques	5		
Basics of Counting	7		
Graphs and Trees	8		
Discrete Probability	5		

Human Computer Interaction KA		Information Management KA	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Foundations	3	Information Management Concepts	5
Designing Interaction	3	Database Systems	5
		Data Modeling	3

Networking and Communications KA		Operating Systems KA	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Introduction	3	Overview of Operating Systems	3
Networked Applications	5	Operating System Principles	3
		Concurrency	1
		Memory Management	2
		Security and Protection	2
		Virtual Machines	2
Parallel and Distributed Computing KA		**Platform-based Development KA**	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Parallelism Fundamentals	3	No Specified KUs	0
Communication and Coordination	1		
Cloud Computing	1		
Programming Languages KA		**Software Development Fundamentals KA**	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Object-Oriented Programming	4	Algorithms and Design	3
Functional Programming	2	Fundamental Programming Concepts	5
Event-Driven and Reactive Programming	2	Fundamental Data Structures	2
Basic Type Systems	2	Development Methods	8

Software Engineering KA		Systems Fundamentals KA	
Knowledge Unit	**Number of Student Learning Outcomes**	**Knowledge Unit**	**Number of Student Learning Outcomes**
Software Processes	3	Computational Paradigms	3
Software Project Management	2	Cross-Layer Communications	3
Tools and Environments	1	Parallelism	3
Requirements Engineering	1		
Software Design	8		
Software Construction	1		
Software Verification and Validation	3		

Social Issues & Professional Practice KA	
Knowledge Unit	**Number of Student Learning Outcomes**
Social Context	4
Analytical Tools	2
Professional Ethics	3
Intellectual Property	2
Privacy and Civil Liberties	4
Professional Communication	3
Sustainability	1
Security Policies, Laws and Computer Crime	3

Correlating Programs to the Computer Science Transfer Curriculum

Examples of programs that align with the ACM Computer Science Transfer Curriculum demonstrate the adaptability to a variety of computer science courses, certificates, and degree programs. Existing correlations from the following colleges can be viewed at the ACM CCECC website: http://ccecc.acm.org/correlations/all.

- Bluegrass Community and Technical College, KY
  - Associate in Science degree, Computer Science
  - Associate in Science degree, Informatics
- El Paso Community College, TX
  - Associate in Arts degree, Computer Science
- Union County College, NJ
  - Associate in Science degree, Computer Science
  - Associate in Science degree, Cybersecurity
- Others

Additional correlations are encouraged. If your college is interested in correlating its certificate and/or degree program courses, visit http://ccecc.acm.org/correlations/new.

Mapping CS Transfer to Other Curricula and Frameworks

The student learning outcomes that make up the present guidance have been mapped to several well-known curricula and frameworks. Current mappings are available for viewing at the ACM CCECC website: http://ccecc.acm.org/guidance.

- ACM/IEEE CS2013
- NSA/DHS CAE2Y Knowledge Units
- College Board AP Computer Science A
- Others

Bibliography

ACM. (1992). *ACM Code of Ethics and Professional Conduct.* New York: Association for Computing Machinery. Retrieved June 15, 2017, from www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct.

ACM and IEEE-CS. (2013). *Computer Science 2013: Curriculum Guidelines for Undergraduate Programs in Computer Science.* New York: Association for Computing Machinery. Retrieved June 15, 2017, from www.CS2013.org.

ACM Two-Year College Computing Committee. (1993). *Computing Curricula Guidelines for Associate-Degree Programs: Computing Sciences.* New York City: ACM Press.

ACM Two-Year College Computing Committee. (2003). *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science.* New York City: ACM Press.

ACM Two-Year College Computing Committee. (2009). *Computing Curricula 2009: Guidelines for Associate-Degree Transfer Curriculum in Computer Science.* New York City: ACM Press.

ACM/IEEE/AIS/IFIP Joint Task Force. (2017). *Cybersecurity Curricula 2017: Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity.* New York City. Retrieved June 15, 2017, from www.csec2017.org.

ACM/IEEE-CS Joint Curriculum Task Force. (1991). *Computing Curricula 1991.* New York City: ACM Press and IEEE Computer Society Press.

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

American Association of Community Colleges. *Partnership Initiatives*. Retrieved June 15, 2017, from International Programs and Services: http://www.aacc.nche.edu/Resources/aaccprograms/international/pi/Pages/default.aspx.

Anderson, L., & Krathwohl, D. (2000). *A Taxonomy for Learning, Teaching, and Assessing: A Revision to Bloom's Taxonomy of Educational Objectives.* New York: Pearson.

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

IEEE Computer Society. Retrieved June 15, 2017, from www.computer.org/education/.

Ministry of Education of the People's Republic of China. Retrieved June 15, 2017, from http://en.moe.gov.cn/.

National Institute of Standards and Technology. (2016). *The National Cybersecurity Workforce Framework.* National Initiative for Cybersecurity Education (NICE). Gaithersburg, MD: NIST. Retrieved June 16, 2017 from www.nist.gov/itl/applied-cybersecurity/nice/resources/nice-cybersecurity-workforce-framework.

NSA and Department of Homeland Security. *CAE2Y Requirements.* Retrieved June 15, 2017, from National IA Education and Training Programs: https://www.iad.gov/NIETP/documents/Requirements/CAE_CD-2Y_Criteria.pdf.

The White House. (2016, February 9). *FACT SHEET: Cybersecurity National Action Plan.* Retrieved June 16, 2017, from Office of the Press Secretary: https://obamawhitehouse.archives.gov/the-press-office/2016/02/09/fact-sheet-cybersecurity-national-action-plan.

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 after 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 composed 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 composed 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 in the context of adversaries. 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." (csec2017.org, June 2017.)

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 to 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.

Appendix A

Cybersecurity-related Student Learning Outcomes in the Computer Science Transfer Curriculum

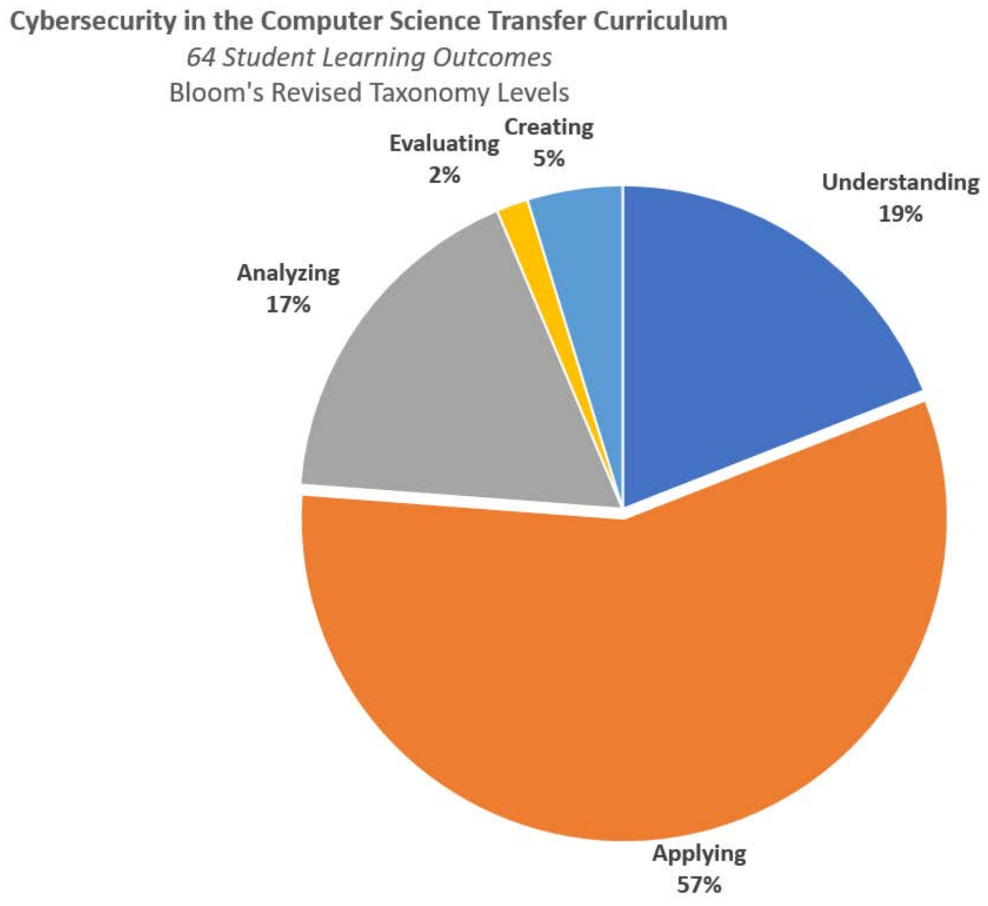

Figure 18: Pie Chart of Bloom's Revised Taxonomy Levels: All Cybersecurity LOs

A total of 214 student learning outcomes are included in the computer science transfer curriculum. Of the 214, there are 64 learning outcomes directly related to cybersecurity, nearly 30%. This Appendix contains the consolidated list of all 64 student learning outcomes carefully woven throughout the transfer BoK. Figure 18 depicts 19% of the cyber outcomes at the *Understanding* level, 57% at the *Applying* level, 17% at the *Analyzing* level, 2% at the *Evaluating* level, and 5% at the *Creating* level of Bloom's Revised Taxonomy. There is clearly an emphasis on doing—the students' ability to apply knowledge—as well as on the higher order thinking skills of analyzing, evaluating, and creating.

AL/Algorithmic Strategies

❖ AL-06. Investigate the use of random/pseudo random number generation in cybersecurity applications.

AL/Fundamental Data Structures and Algorithms

❖ AL-14. Investigate security vulnerabilities in various data structures.

AR/Machine Level Representation of Data
- ❖ AR-03. Illustrate how fixed-length number representations could affect accuracy and precision, causing vulnerabilities.

CYB/Foundational Concepts in Security
- ❖ CYB-01. Describe security as a continuous process of tradeoffs, balancing between protection mechanisms and availability.
- ❖ CYB-02. Illustrate through examples the concepts of risk, threats, vulnerabilities, attack vectors, and exploits, noting there is no such thing as perfect security.
- ❖ CYB-03. Investigate various countermeasures and security controls to minimize risk and exposure.
- ❖ CYB-04. Analyze the tradeoffs of balancing key security properties, including Confidentiality, Integrity, and Availability (CIA).
- ❖ CYB-05. Explain the concepts of trust and trustworthiness related to cybersecurity.
- ❖ CYB-06. Examine ethical issues related to cybersecurity.
- ❖ CYB-07. Illustrate various ways to minimize privacy risks and maximize anonymity.
- ❖ CYB-08. Apply security principles and practices in a dynamic environment.
- ❖ CYB-09. Illustrate through examples the key role risk management frameworks play in identifying, assessing, prioritizing, and controlling risks to organizational assets.
- ❖ CYB-10. Illustrate with examples the goals of end-to-end data security.

CYB/Principles of Secure Design
- ❖ CYB-11. Use the principles of secure design.
- ❖ CYB-12. Illustrate the security implications of relying on open design vs the secrecy of design.
- ❖ CYB-13. Discuss the benefits and limitations of designing multiple layers of defenses.
- ❖ CYB-14. Analyze the tradeoffs associated with designing security into a product.

CYB/Defensive Programming
- ❖ CYB-15. Construct input validation and data sanitization in applications, considering adversarial control of the input channel.
- ❖ CYB-16. Explain the tradeoffs of developing a program in a type-safe language.
- ❖ CYB-17. Implement programs that properly handle exceptions and error conditions.
- ❖ CYB-18. Examine the need to update software to fix security vulnerabilities.

CYB/Threats and Attacks
- ❖ CYB-19. Examine likely attack types against standalone and networked systems.
- ❖ CYB-20. Illustrate the key principles of social engineering, including membership and trust.

CYB/Cryptography
- ❖ CYB-21. Describe key terms in cryptology, including cryptography, cryptanalysis, cipher, cryptographic algorithm, and public key infrastructure.
- ❖ CYB-22. Use a variety of ciphers to encrypt plaintext into ciphertext.
- ❖ CYB-23. Apply cryptographic hash functions for authentication and data integrity.
- ❖ CYB-24. Contrast symmetric and asymmetric encryption in relation to securing electronic communications and transactions.

CYB/Web Security
- ❖ CYB-25. Explain browser and web security model concepts, including same-origin policy, web sessions, and secure communication channels.

GV/Fundamental Concepts
- ❖ GV-05. Perform information hiding through steganography in images, messages, videos, or other media files.

HCI/Designing Interaction
- ❖ HCI-03. Examine the issues of trust in HCI, including examples of both high and low trust systems.
- ❖ HCI-06. Analyze the interaction between a security mechanism and its usability.

IM/Information Management Concepts
- ❖ IM-03. Investigate contingency planning with respect to business continuity, disaster recovery, and incident response.
- ❖ IM-04. Describe proven techniques to secure data and information.

IM/Database Systems
- ❖ IM-10. Investigate vulnerabilities and failure scenarios in database systems.

IM/Data Modeling
- ❖ IM-12. Diagram a relational data model for a given scenario that addresses security and privacy concerns.

NC/Networked Applications
- ❖ NC-06. Describe security concerns in designing applications for use over wireless networks.
- ❖ NC-07. Illustrate secure connectivity among networked applications.

OS/Overview of Operating Systems
- ❖ OS-03. Discuss potential threats to operating systems and the security features designed to guard against them.

OS/Security and Protection
- ❖ OS-10. Investigate the features and limitations of an operating system used to provide protection and security.
- ❖ OS-11. Use mechanisms available in an operating system to control access to resources.

PD/Communication and Coordination
- ❖ PD-04. Implement mutual exclusion to avoid race conditions that could cause security vulnerabilities.

PD/Cloud Computing
- ❖ PD-05. Investigate the challenges and concerns related to security and privacy in Cloud computing.

PL/Object-Oriented Programming
- ❖ PL-03. Use access modifiers to secure class data and methods.

PL/Event-Driven and Reactive Programming
- ❖ PL-08. Describe potential security vulnerabilities in event-driven GUI applications.

PL/Basic Type Systems
- ❖ PL-10. Explain the security implications of a type-safe language for software development

SDF/Fundamental Programming Concepts

- SDF-05. Investigate potential vulnerabilities in provided programming code.
- SDF-06. Create programs which use defensive programming techniques, including input validation, type checking, and protection against buffer overflow.

SDF/Development Methods

- SDF-12. Investigate common coding errors that introduce security vulnerabilities.
- SDF-19. Carry out a code review on a program component using a provided security checklist.

SE/Software Processes

- SE-03. Diagram the phases of the secure software development lifecycle (SecSDLC).

SE/Software Project Management

- SE-05. Investigate the risks in using third- party applications, software tools, and libraries.

SE/Requirements Engineering

- SE-07. Implement the requirements for a secure software system.

SE/Software Design

- SE-09. Analyze an existing software design to improve its security.
- SE-10. Describe the cost and tradeoffs associated with designing security into software.

SE/Software Verification and Validation

- SE-14. Design a test plan that validates software security.

SF/Computational Paradigms

- SF-03. Investigate security implications related to emerging computational paradigms.

SF/Cross-Layer Communications

- SF-06. Investigate defects in a layered program using tools for program tracing, single stepping, and debugging.

SP/Social Context

- SP-04. Investigate social engineering attacks and the types of bad actors who might
- perform them.

SP/Professional Ethics

- SP-08. Support the ethical responsibility of ensuring software correctness, reliability, and safety.

SP/Privacy and Civil Liberties

- SP-13. Investigate threats to privacy rights in personally identifiable information (PII).
- SP-15. Analyze technological solutions to privacy concerns.

SP/Security Policies, Laws and Computer Crime

- SP-20. Investigate laws applicable to computer crimes.
- SP-21. Examine the motivation and ramifications of cyber terrorism and criminal hacking.
- SP-22. Write a company-wide security policy.

Appendix B

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.

Table 6 Measurable Action Verbs of Bloom's Revised Taxonomy, Cognitive Domain

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			

CCECC.ACM.org

Association for
Computing Machinery

Advancing Computing as a Science & Profession