Computing Curricula 2009:
Guidelines for Associate-Degree
Transfer Curriculum in Computer Science

ACM
Two-Year College
Education Committee

The ACM Two-Year College Education Committee

Elizabeth K. Hawthorne, Committee Chair
Union County College

Robert D. Campbell
Graduate Center, City University of New York

Karl J. Klee
Alfred State College

Anita Wright
Camden County College

This report is available for viewing and download at: http://www.acmtyc.org/

CONTENTS

CHAPTER 1: OVERVIEW

SECTION 1.1: GLOSSARY OF TERMS

The ACM Two-Year College Education Committee defines the following terms in relationship to curricula associated with computing education in associate-degree granting institutions.

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

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

Computing is now recognized as comprised of five defined sub-disciplines: computer science, computer engineering, software engineering, information systems and information technology.

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.

SECTION 1.2: TWO-YEAR COLLEGE ENVIRONMENT

According to the American Association of Community Colleges, approximately one-half of all undergraduates in the United States are enrolled in community colleges, and more than half of all first-time college freshman attend community colleges. "Community colleges are centers of educational opportunity. They are an American invention that put publicly funded higher education at close-to-home facilities, beginning nearly 100 years ago with Joliet Junior College [in Joliet, Illinois]. Since then, they have been inclusive institutions that welcome all who desire to learn, regardless of wealth, heritage, or previous academic experience. The process of making higher education available to the maximum number of people continues to evolve" (http://www.aacc.nche.edu/).

The two-year 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 or cultural activities.

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

Two-year, community or technical colleges, as well as certain four-year colleges, award associate degrees to students completing two years of study. These associate-degree programs are complete in their own right, whether designed specifically to enable graduates to transfer into the upper division of a baccalaureate program or to gain entry into the workforce. These institutions also offer certificate programs, intended to be fulfilled in less time than a complete degree program; such programs are often designed for targeted student audiences and focused on specific content.

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

SECTION 1.3: ARTICULATION

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

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

Many associate-degree granting institutions have established articulation ("2+2") agreements between their associate degree career programs and corresponding high school programs. For example, many two-year colleges award up to 12 credit hours toward an associate degree to students who complete the Cisco Networking program in high school. Similarly, community

colleges may award up to 12 college credits toward an associate degree for high school students passing the Certified Internet Web Professional examination.

SECTION 1.4: THE SUB-DISCIPLINES OF COMPUTING

The Association for Computing Machinery currently categorizes the overarching discipline of computing into five defined sub-disciplines: computer science, computer engineering, software engineering, information systems and information technology. The ACM Two-Year College Education Committee describes these disciplines as follows:

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.

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

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.

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, ecommerce, finance, customer relations management, transaction processing, and data-driven decision-

making and executive support. Upon graduation, students initiating careers as information systems specialists should be able to analyze information requirements and business processes and be able specify and design systems that are aligned with organizational goals.

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

For each, the ACM Two-Year College Education Committee presents a specific curriculum, with associated commentary about the unique considerations and distinguishing aspects of the program outcomes, teaching and learning strategies and coursework pursuant to the nature of the computing sub-discipline being studied.

SECTION 1.5: ACM COMPUTING ONTOLOGY

The computing sub-disciplines have much in common as well as distinguishing characteristics. The ACM "Ontology Project" was launched in order to organize and identify these commonalities and distinctions in a formal manner; the goals of this project are:

- to represent the entirety of the computing and information-related sub-disciplines together;
- to provide a mechanism for easy update of the information in a timely way;
- to illustrate the differences and the overlaps of the various sub-disciplines that address these topics; and
- to describe fully the various topics and subtopics of interest to educators and researchers in any of the sub-disciplines concerned with computing and the management and processing of information.

The ontology classifiers can be applied at the course-topic level to assist in identifying content commonalities beyond shared course requirements.

SECTION 1.6: BACCALAUREATE PROGRAM GUIDELINES

The professional societies of the ACM, the IEEE Computer Society, the Association of Information Technology Professionals and the Association for Information Systems have a history of collaborating on computing materials for higher education. These organizations have

jointly produced significant volumes of curricular recommendations and guidelines for baccalaureate and graduate computing programs; these volumes are referred to as the ACM Computing Curricula series. Likewise, the ACM Two-Year College Education Committee 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 report provides discussion on transfer considerations and discussion on articulation.

SECTION 1.7: ACCREDITATION

Regional institutional accreditation is common among the associate-degree granting institutions in the United States. Such accreditation is intended to promote public confidence that institutions are maintaining a defined level of educational excellence, as validated by quality assurance and improvement through a rigorous process of peer evaluation. Under this process institutions are assessed and accredited in whole, with each component of the college or university contributing to a shared mission, as evidenced by comprehensive measures of institutional effectiveness and holistic assessment of defined student outcomes. In a university setting, individual colleges or schools (for example, in fields such as business, medicine, dentistry, etc.) may be accredited by associations that specialize at that particular level.

SECTION 1.8: PROGRAM ACCREDITATION

Program accreditation, as distinguished from regional institutional accreditation and school accreditation, has long been available for particular disciplines in associate-degree granting institutions, including programs such as electrical technology, mechanical technology and construction technology (and of course notably in health-related fields). Even associate-degree granting institutions not pursuing specific program accreditation often take pride in the placement and subsequent success of their graduates into the upper division of accredited baccalaureate programs. Curricular guidelines have frequently taken into consideration program accreditation standards. While the relationship between accrediting criteria and curricular guidelines should be a symbiotic one, inasmuch as they have the mutual goal of sound student preparation in the discipline, the criteria for program accreditation do not necessarily directly impact curricular guidelines given that the accrediting criteria and curricular guidelines arise from differing sources. Accreditation is also closely associated with assessment.

Sources of information on accreditation include the United States Department of Education, the Council for Higher Education Accreditation, and the American Association of Community Colleges. Some specific agencies accrediting associate-degree programs include:

- ABET, formerly the "Accreditation Board for Engineering and Technology", which through its Computing Accreditation Commission and Technology Accreditation Commission accredits postsecondary degree-granting programs housed within regionally accredited institutions.
- The Accrediting Commission of Career Schools and Colleges of Technology, which promotes the development of a highly trained and competitive workforce through quality career oriented education.
- The Accrediting Council of Independent Colleges and Schools, which accredits programs designed to educate students for professional, technical, or occupational careers, including those that offer those programs via distance education.
- The National Association of Industrial Technology, which accredits industrial technology programs in colleges, universities, and technical institutes.

SECTION 1.9: TRANSFER PROGRAMS

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

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

It is not uncommon for students to complete an associate-degree program of study, choose to work for a period of time, and then return to college to pursue their upper division studies for

career advancement. (And many employers will provide tuition reimbursement for workers who wish to continue toward a baccalaureate degree.) Because of the ever evolving nature of computing, students must be aware that course content and program requirements are updated frequently, potentially subjecting them to new program requirements and revised articulation agreements. Students are best served when sequences of courses are completed as a unit at one institution due to the comprehensive and conceptual nature of the computing and mathematics content. Hence, students should complete programs of study in their entirety up to well-defined exit points at one institution before transferring to another institution; articulation cannot be expected to accommodate potential transfers in the middle of a well-defined and recognized body of knowledge. Therefore, the ACM Two-Year College Education Committee strongly recommends that the entire CS I – CS II – CS III core course sequence be completed at the same educational institution.

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, computer engineering, software engineering, information systems and information technology.

SECTION 1.10: 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 work force directly upon graduation.

Career-oriented associate-degree programs provide students with the specific knowledge, skills and abilities 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. In addition, many students will augment their formal studies with technical certifications to enhance their immediate employability.

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

- An active industry advisory committee consisting of prospective employers, providing guidance concerning the knowledge, skills, and abilities students must possess to enter directly into a career within their community.
- Real-world work experience including co-op programs, internships and other practicum activities, with an emphasis on professional practices.
- Core and elective coursework as recommended by advisory committees.
- Integration of technical, communication and time-management skills, team projects, and other interpersonal skills that prepare the student for a business working environment.
- Potential articulation paths that enable the career-oriented student to pursue a baccalaureate degree in the future after working for some period of time.
- Assessment processes whereby students can earn credit for relevant experience.

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

The ACM, through its Two-Year College Education Committee, provides curricular resources for associate-degree career-oriented programs in a variety of computing disciplines.

SECTION 1.11: ETHICS AND PROFESSIONALISM

Professional, legal and ethical issues are important elements in the overall curricula for computing disciplines, and must be integrated throughout the programs of study. This context should be established at the onset and these matters should appear routinely in discussions and learning activities throughout the curriculum. The ACM Code of Ethics notes that "When designing or implementing systems, computing professionals must attempt to ensure that the products of their efforts will be used in socially responsible ways, will meet social needs, and will avoid harmful effects to health and welfare." The Code goes on to provide an excellent framework for conduct that should be fostered beginning early in students' experiences.

As computing technologies become ubiquitous in society, ethical behavior and adherence to codes of conduct for computing professionals is imperative; therefore, careful consideration of

legal, ethical, and societal issues involving computing, the Internet and databases are essential to the education of computing professionals. Students who realize the potential uses and abuses of technology will, as citizens, be able to contribute to public policy debate from a knowledgeable perspective on issues such as property rights and privacy concerns that affect everyone.

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

SECTION 1.12: SECURITY IN THE COMPUTING CURRICULA

Whether referred to as “computer security”, “information security”, “information assurance” or some other heading, curriculum content in creating and maintaining secure computing environments is a critical component in all 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 build and configure secure software systems from the bottom up; web specialists must be capable of assessing risks and determining how best to reduce the potential impact of breached systems; user support technicians must be knowledgeable in security concerns surrounding desktop computing; and project managers must be able to calculate the cost/benefit tradeoffs involved with implementing secure systems.

It is the responsibility of faculty to ensure that students are well prepared for the security challenges they will inevitably encounter in their careers as computing professionals. This can be addressed by way of a variety of implementation strategies. One approach that some associate-degree computing programs offer is a host of individual courses on specific security topics. This approach can provide a wealth of content opportunities for specialization but may

create scheduling challenges for many students as it runs the risk of students graduating without having taken sufficient electives to achieve the understanding of the security concepts necessary to function in their professional roles. Another approach is to fully integrate and incorporate fundamental security topics into core computing courses in the program of study with specialized courses reserved for targeted settings; this is the approach promoted by the ACM Two-Year College Education Committee in its computing curriculum resources. The Committee also advocates strongly for learning activities that require students to actively demonstrate mastery of the tenets of professional conduct, ethical and responsible behavior and appreciation for security matters in a holistic manner.

SECTION 1.13: CORE COMPUTING COURSES

Among the associate-degree transfer programs, the Computer Science I-II-III coursework provides a common "core" body of knowledge in computing. The complete course sequence is designed in such a manner that students progress in knowledge, proficiency and professional maturity in several specific areas, including software engineering principles and professional and ethical conduct.

The progression of software engineering topics across CS I-II-III originates in CS I, where there is an emphasis on using a cyclic approach for program development by iterating through designing, coding, and testing program modules. Complemented by algorithm analysis, students are encouraged to think abstractly about problems and to begin developing processes for decomposing problems into organized parts. Encouraging clear documentation, good naming conventions and consistent secure coding style contribute to a disciplined approach to writing programs.

The progression of software engineering topics across CS I-II-III continues in CS II, where greater emphasis is placed on abstraction and sound software design principles, engaging students in the development of secure software components that solve a wide range of related problems and can be reused. The students determine the necessary elements of simple ADTs (such as a counter or a date) and then construct them; by their very nature, these components must be well-documented to encourage reuse. Additionally the students write assertions such as pre-conditions and post-conditions describing each class method, thereby encouraging students to think deeply about a simple problem before coding. After coding, the components must be well-tested, and therefore the use of test plans and test drivers are practiced. These activities reinforce the notion of constructing software from well-defined, independent pieces and complement the study of using existing library classes and APIs in software solutions.

The progression of software engineering topics across CS I-II-III concludes in CS III, where students are asked to step beyond the programmer role and take a broader view of software development; to consider its lifecycle from problem description to maintenance. Students first practice with analysis and design of medium-sized systems. Standard modeling tools are introduced and the students complete the phases of analysis, design, implementation and testing of a medium-sized team project that includes documents such as UML diagrams or CRC cards in addition to test plans. Students consider design patterns and write applications using data structures and templates. The software engineering topics are integrated with professionalism and ethics, as well as software and information assurance topics, such as security concerns and liabilities of computer-based systems.

The progression of the emphasis on professional and ethical conduct across CS I-II-III originates in CS I, where the curriculum is designed to consider the historical context of computing and programming as well as examining issues involving ethical conduct; plagiarism, intellectual property and software piracy issues are presented. Typically, students' requirements for submitting original work as well as college policies regarding the use of computing resources and acceptable computing behavior on campus and the Brookings Institute "10 Commandments of Ethical Computing" can be used as relevant discussion starters.

The progression of the emphasis on professional and ethical conduct across CS I-II-III continues in CS II, which builds upon this foundation by examining societal issues, the Internet, and professionalism. Now that the students have gained some experience with developing programs, they can begin to see "what can go wrong" and the possible consequences to the user of their program, at a personal level, such as infinite loops and program crashes. Additionally, students are confronted with broader implications by design considerations regarding databases and data accessibility; ethical concerns regarding personal data, privacy and property rights should be explored Integrating these topics with the software development process, security issues, and relevant cases of software errors will help students recognize that their work can have individual as well as societal consequences and encourage them to think carefully about the design and implementation of their programs.

The progression of the emphasis on professional and ethical conduct across CS I-II-III concludes in CS III, where a broader view is presented – encompassing computing sciences as a profession. Standards of professional behavior, organizations and publications are examined as well as a variety of occupational roles in the computing field. Course content materials presenting case studies of significant software failures amplify the topics of risks and liabilities. The students should start to recognize that invariably software production involves ethical choices. Incorporating these topics with the software lifecycle, engineering, human factors, and software assurance considerations will help students internalize the significance of professional

and ethical behavior and subsequently demonstrate it through their individual and group projects.

These progressions can be summarized based on course content Topic Headings and can also be mapped to the ACM Computing Ontology Topic Classifiers used in the computing ontology.

CS Core Sequence Topics

CS I Topic Headings	**CS II Topic Headings**	**CS III Topic Headings**	**ACM Computing Ontology Topic Classifiers**
Social and historical context of computing	Ethical conduct	Professionalism	Ethical Social; History Computing
Programming languages	Event-driven programming		Programming Languages
IDE and software tools			Programming Languages
Fundamental programming constructs	Intermediate programming constructs	Recursion	Programming Fundamentals; Programming Languages
Machine level representation of data			Computer Hardware Organization
Fundamental algorithms & problem-solving	Intermediate computing algorithms	Formal computing algorithms	Algorithms Complexity; Discrete Structures
	Object-oriented design & modeling	Software reuse	Conceptual Modeling; Information Topics
Object-oriented principles	Object-oriented programming		Programming Languages;
Fundamental Data Structures	Intermediate data structures	Canonical data structures	Programming Languages; Algorithms Complexity;
Secure code	Software assurance	Software and information assurance	Security Topics
Overview of operating systems			Computing & Network Systems

Human-computer interaction	Human-computer interaction	Human-computer interaction	User Interface; Graphics, Visualization, Multimedia
	Simple database integration		Information Topics
Program development	Software development	Software engineering	Software Engineering
		Basic algorithmic analysis	Algorithms Complexity
		Algorithmic strategies	Algorithms Complexity

SECTION 1.14: ASSESSMENT

Institutions must ensure an ongoing and effective process for assessing student learning. In particular, computing courses and programs of study must incorporate clearly defined, measurable student outcomes which demonstrate that student achievement at the course level promotes successful attainment of program goals.

This relationship is demonstrated when:

- for each program of study a collection of program outcomes is identified;
- for each course in the program a collection of student learning outcomes is identified;
- for each course, topics of study and learning activities are selected and designed to support the course student outcomes;
- each course student outcome supports one or more program outcomes; and
- each program outcome is supported by one or more course outcome.

Effective assessment provides valuable feedback to faculty and academic leaders for continuous improvement of pedagogy, course content and program outcomes, in order to better prepare students for future studies and careers. In addition, effective assessment fosters articulation between institutions and promotes student transfer, and documents employment readiness and facilitates job placement. Accreditation requirements, performance-based funding and public demands for accountability also make effective educational assessment a necessity.

Avenues for assessing the success of an associate-degree program might include:

- institutional or program accreditation
- student performance-based measurements

- industry advisory councils
- program completion rates for students
- job placement rates for students
- post-transfer success by students
- student success rates on certifying examinations

For an in-depth tutorial of program assessment see http://online.bc.cc.ca.us/courseassessment/.

SECTION 1.15: TEACHING AND LEARNING STRATEGIES

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

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

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

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

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

SECTION 1.16: COMPUTING LABORATORY EXPERIENCES

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

SECTION 1.17: GENERAL EDUCATION

Associate-degree programs are subject to general education requirements which mandate that students complete a minimum number of courses spanning a variety of major discipline categories. Such requirements vary among states, institutions and the type of associate degree being pursued (e.g., AAS, AA, or AS). Colleges must ensure that degree programs include the courses appropriate to fulfill all general education and related requirements arising from institutional and state mandates, as well as regional institutional accreditation guidelines. Categories of general education courses typically include: mathematics and quantitative reasoning; natural sciences; English and oral communication; cultural and diversity studies, humanities and the arts; world languages; social and behavioral sciences; health and physical education.

Programs best serve the students by requiring them to take general education courses that provide a social context for their overall education. Today's world is one of rapid change requiring routine interaction on a global scale with individuals of diverse cultures and languages, and placing a much greater emphasis on interpersonal skills. Students should engage courses as part of their overall program of study that assist them in preparing for this world; in like fashion, such perspectives should be widely incorporated into numerous courses across many disciplines.

Effective abilities in oral and written communication are of critical importance to computing professionals; these skills must be established, nurtured and incorporated throughout a computing curriculum. Students must master reading, writing, speaking, and listening abilities, and then consistently demonstrate those abilities in a variety of settings: formal and informal, large group and one-on-one, technical and non-technical, point and counter-point. Many of the skills found in a technical writing course often benefit a computing curriculum (these include learning to write clearly and concisely; researching a topic; composing instructions, proposals, and reports; shaping a message for a particular audience; and creating visuals). Overall, student learning activities should span the curriculum and should include producing technical writing and report writing, engaging in oral presentations and listening activities, extracting information from technical documents, working in a group dynamic, and utilizing electronic media and modern communication techniques.

SECTION 1.18: MATHEMATICS

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.

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 Discrete Structures course described herein 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 data structures and algorithms in the CS I - CS II - CS III course sequence. This foundational 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. The discrete structures concepts 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 the calculus are required for the study of efficiency of algorithms and the notion of Big-O; for that reason the Calculus I course must be completed concurrently with CS III and the study of data structures. The Calculus I course described herein includes the foundational concepts of limits, functions, and upper and lower bounds necessary to understanding asymptotic analysis. 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 the calculus. The principal strategies in software development – formal approaches to solving problems and reusable techniques – underlie the calculus coursework as well.

SECTION 1.19: NATURAL SCIENCES

Rigorous laboratory science courses such as physics, chemistry and biology provide students pursuing associate-degree transfer programs in computing with content knowledge, direct hands-on laboratory experiences and strong training in the tenets of the “scientific method”. The scientific method (summarized as formulating problem statements and hypothesizing, designing and conducting experiments, observing and collecting data, analyzing and reasoning, and evaluating and concluding) reasonably presents a basic 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 computing courses should be designed to incorporate and reinforce this framework. 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. Science courses can provide important content for distinct specializations within computing disciplines; such considerations will vary by institution based on program design and resources. Program requirements of this nature can provide students with a crucial

foundation should they later pursue computing careers in those scientific domains (for example, bioinformatics).

SECTION 1.20: COMPUTING FOR OTHER DISCIPLINES

Computing technology has become an integral part of every field of study and every profession. Therefore, an institution must consider the means by which its computing curricula can also be responsive to the needs of "computing for other disciplines". The groundbreaking 1993 report Computing Curricula Guidelines for Associate Degree Programs in Computing for Other Disciplines, produced by the ACM Two-Year College Education Committee, provided the first formal set of recommendations identifying courses that a computing department could offer for students in other disciplines.

As our information-oriented and technology-enabled society has moved forward, the need for individuals in fields outside computing to learn not only about the application of technology but also about its fundamental principles has increased dramatically. From managing and mining large collections of intellectual property to developing enhanced consumer products to creating works of art, ever-increasing access to computing power and facility with applications software can still fall short without knowledge about underlying systems and insights into the foundational concepts involved.

No single prescription can be given for a set of courses to meet the computing needs of students in other disciplines. Mathematics majors may be interested in learning about computer programming; nursing majors may be interested in information security; and engineering majors may have a need to learn about the Linux operating system. However, computing faculty can identify courses in their computing curricula that may be accessible – with the necessary prerequisite requirements fulfilled – and may be suitable – based on course design and student learning outcomes – for students in other fields and settings. The courses suitable for computing for other disciplines are noted in the ACM Two-Year College Education Committee computing course inventory.

SECTION 1.21: INTERNATIONALIZATION

The ACM Two-Year College Education Committee computing curricula resources provide meaningful guidance for two-year post-secondary ("tertiary") higher education programs both locally within the United States and globally throughout the world.

The American Association of Community Colleges (AACC) has established an Office of International Programs and Services whose stated goals are "to advocate the community

college role in global education among key constituencies, nationally and internationally, to advance global exchanges and partnerships between member colleges and international entities, and to promote intercultural understanding and engagement among students, faculty, staff, and decision makers." James McKenney, AACC Vice-President for Economic Development and International Programs, spoke as early as 2002 in a reflective interview titled "The Global Linkage" appearing in the journal "US Society and Values" about the rapidly expanding phenomenon of "community colleges", "two-year technical colleges", and the "two-year structure" in general throughout Europe, the Americas and Asia.

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

Clearly then curricular guidance for computing programs found in the first two years of a post-secondary higher education setting has the potential for global impact. The ACM Two-Year College Education Committee computing curricula resources provide a strong foundation for such programs, by defining specific content in a structured format, describing meaningful pedagogy and measureable student learning outcomes, and detailing rubrics for effective assessment of student learning. While some specific implementation aspects of these resources may be more relevant or prominent in the United States the resources are useful and sound, readily applicable to numerous settings and easily adapted to a wide variety of implementation strategies.

CHAPTER 2: PROGRAM INVENTORY

SECTION 2.1: COMPUTER SCIENCE

Program Title: Computer Science
Program Type: Transfer
Program Baccalaureate Report(s): http://www.acm.org/education/curricula-recommendations
Program Overview
The foundation for the Computer Science associate-degree transfer program is the three-course computing sequence CS I - CS II - CS III. This sequence should be accompanied by the opportunity for additional computing courses based on a variety of factors, including transfer requirements, institutional specializations, and student interests.

Past computer science model curricula have identified at least three "paradigms" or approaches that one could take to computer science content: objects-first (centered on object-oriented programming), breadth-first (an initial holistic view subsequently progressing deeper), and imperative-first (centered on procedural programming). The Computer Science associate-degree transfer program now calls for a blended approach:

- Object-oriented programming is emphasized in CS I, but not necessarily early in the semester.
- The topics of algorithms and fundamental programming constructs are important components of CS I and are consistent with the Böhm-Jacopini theory for procedural programming.
- The breadth-first approach is used in the coverage of three important topics: ethics and professionalism, security, and software engineering principles.

These topics are covered in ever-deeper fashion as the student progresses through CS I-II-III.

Course content on software engineering principles is prominent in the Computer Science associate-degree transfer curriculum. An essential tenet of software engineering is ensuring a disciplined, controlled approach to software evolution and reuse. The principles of software engineering progress in the CS I-II-III course sequence from an initial focus on program development to software development and then to broader software engineering concerns.

Security topics are covered in deeper and deeper fashion as the student progresses from CSI to CSII to CS III. In CS I, students use encapsulation to incorporate privacy into their applications. In CS II, students use security-aware exception handling to help prevent buffer overflows, memory leaks and back-door accesses. In CS III, students develop and ensure robust attack-resistant code by testing applications for known security flaws.

Similarly, the course content across CS I-II-III forms a natural progression for the topics of social awareness, ethics, privacy and legal concerns, and professionalism. The first course (CS I) begins by examining the historical context of computing and ethical conduct, focusing on individual behaviors. The second course (CS II) follows by considering the societal impacts of computing and the Internet and encouraging students to recognize the direct and far-reaching effects of their work and behaviors. Culminating in the third course (CS III), the students begin internalizing the importance of professional and ethical behavior through their project activities, team interactions, and exposure to professional organizations and publications; the goal is to help the students begin to view themselves as ethical software developers.

In addition to the CS I-II-III core sequence, a collection of additional "intermediate" courses are identified providing a variety of paths of study and serving local considerations. These courses also add specific content to the Computer Science program of study, including a focused emphasis on security (the Essentials of Computing Security course), preparation for future study in software engineering (the Introduction to Software Engineering course), and – importantly – an opportunity early on (via the one-credit Survey of Computing Disciplines course) for students to make informed decisions about their anticipated career plans and the selection of a curriculum to pursue.

The theory of concurrent programming and synchronization, as it relates to operating systems (mutual exclusion, semaphores, deadlock, race conditions), is covered in the Computer Organization and Architecture course. The application of concurrency in programming (threads) is covered in the Computer Science III course. The theory of running threads concurrently using multiple processor cores or concurrently on parallel computers is covered in the Hardware Fundamentals course and the Programming Languages course.

Students are best served by completing the three-course sequence CS I - CS II - CS III, together with additional intermediate computing courses to be determined by the local institution based on the requirements of transfer institutions, expertise of the faculty, and availability of hardware and software. With these additional intermediate computing courses, students will be better prepared to transfer successfully into the upper division of a baccalaureate degree program and will have acquired a reasonable level of understanding in the various subject areas that define the discipline, as well as develop an appreciation for the interrelationships among these areas.

The Computer Science associate-degree transfer program includes a minimum of two terms of mathematics preparation: Discrete Structures and Calculus I; articulation agreements between baccalaureate institutions and associate-degree institutions may define additional mathematics requirements. It is often the case that students who enter a two-year college intending to pursue computer science have insufficient mathematics preparation; such students should be

counseled to complete a rigorous pre-calculus course in their first semester of study. This preparation will enable these students to engage the CS I course simultaneously with their mathematics studies, and to remain on schedule to graduate in four semesters.

The Computer Science associate-degree transfer curriculum represents the program course configuration appropriate to the United States. In the first term (typically a semester, but in alternative learning environments perhaps a different timeframe), students begin their computer science core sequence, have the opportunity to prepare themselves (as needed) mathematically, and address general education requirements. In terms two and three, students continue the computer science core sequence, complete the required mathematics courses, and continue addressing general education coursework. Terms three and four provide an opportunity for intermediate computer science courses or additional mathematics preparation; such courses must be selected based on student interests, transfer requirements and articulation agreements.

Program Outcomes

Student learning outcomes that are clearly defined and effectively assessed are essential at the course level to ensure that students are progressing meaningfully through a program of study. Similarly, well-defined student outcomes are crucial at the program level to ensure that graduates are well equipped to master coursework upon transfer to the upper division. Furthermore, well defined course and program outcomes are essential tools in developing effective articulation agreements.

Group 1: Critical Thinking, Problem Solving, and Theoretical Foundations

Upon successful completion of the Computer Science associate-degree program, a student will have demonstrated:

A. An ability to apply knowledge of computing and mathematics appropriate to the discipline.
B. An ability to think critically and apply the scientific method.
C. An ability to analyze a problem and craft an appropriate algorithmic solution.
D. An ability to design, implement and evaluate an appropriate and secure computer-based system, process, component, or program to satisfy required specifications.

Group 2: Communication and Interpersonal Skills

Upon successful completion of the Computer Science associate-degree program, a student will have demonstrated:

A. An ability to read and interpret technical information, as well as listen effectively to, communicate orally with, and write clearly for a wide range of audiences.
B. An ability to function effectively as a member of a team to accomplish common goals.

Group 3: Professionalism and Ethics, Social Awareness and Global Perspective

Upon successful completion of the Computer Science associate-degree program, a student will have demonstrated:

A. An ability to engage in continuous learning as well as research and assess new ideas and information to provide the capabilities for lifelong learning.
B. An ability to exhibit professional, legal and ethical behavior.
C. An ability to demonstrate social awareness, respect for privacy and responsible conduct.
D. An ability to analyze the global impact of computing on individuals, organizations, and society.

These program outcomes can be summarized as shown below. For each outcome, the table below also identifies the underlying support provided by the CS I-II-III core sequence and by the foundational mathematics courses called for in the curriculum.

Program Outcomes and Supporting Coursework

Program Outcomes	**Group I Critical Thinking, Problem Solving, and Theoretical Foundations**				**Group 2 Communication and Interpersonal Skills**		**Group 3 Professionalism and Ethics, Social Awareness and Global Perspective**			
Program Learning Outcome	**1. A**	**1.B**	**1.C**	**1.D**	**2.A**	**2.B**	**3.A**	**3.B**	**3.C**	**3.D**
CS I	X	X	X	X	X		X		X	X
CS II	X	X	X	X	X	X	X		X	X
CS III	X	X	X	X	X	X	X	X	X	X
Discrete Structures	X	X	X		X		X			
Calculus I	X	X	X				X			

Program Coursework

The coursework called for by these guidelines includes a core CS sequence, as well as foundational mathematics content. These courses have fundamental sequential (pre-requisite) relationships as well as parallel (co-requisite) relationships. These relationships can be summarized as noted in the table below.

Foundational coursework sequencing

	Term 1	Term 2	Term 3	Term 4
COMPUTER SCIENCE COURSES	Computer Science I	Computer Science II	Computer Science III	
MATHEMATICS COURSES	Pre-calculus (if needed)	Discrete Structures	Calculus I	

In addition to the core foundational coursework, the curriculum must include general education courses and support (or "elective") courses. These courses are outlined in a typical four-semester schedule as noted in the table below.

Program Course Sequencing

Semester 1	Semester 2
Computer Science I	Computer Science II
Pre-Calculus	Discrete Structures
Survey of Computing Disciplines	General Education
General Education	General Education
English I	English II
Semester 3	**Semester 4**
Computer Science III	Computer Science course
Calculus I	Mathematics course Computer Science course
Computer Science course Engineering Course	Mathematics course Computer Science course Engineering Course
Natural Science I	Natural Science II
General Education	General Education

The non-specified courses in Computer Science and Engineering should be selected from the computing and engineering courses appearing in the Course Inventory in Chapter 3.

CHAPTER 3: COURSE INVENTORY

SECTION 3.1: COMPUTING COURSES

Course Title: Computer Science I

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

Course Prerequisite(s):

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

Course Co-requisite(s):

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

Course Type: Computer Course

Eligible for Computing for Other Disciplines? Yes

Course Topics:

- Historical context of computing (1hour): history of computing ideas, computing, and programming;
- Ethical conduct (1 hour): codes of ethics and responsible conduct; intellectual property, copyright, and plagiarism; "Ten Commandments for Computer Ethics"
- Programming languages (1 hour): comparison of object-oriented, procedural, functional programming
- Software tools and IDE (2 hours): compiling, interpreting, linking, executing, testing and debugging
- Fundamental programming constructs (11 hours): basic syntax and semantics of a higher-level language; variables (scope and lifetime), types, expressions, and assignment; self-documentation; standard and file I/O; conditional and iterative control structures; structured decomposition; pseudo-random number generator
- Machine level representation of data (1 hours): overview of the storage of instructions, numbers and characters in a Von Neumann machine
- Fundamental algorithms and problem-solving (6 hours): problem-solving strategies; the role of algorithms in the problem-solving process; implementation strategies for algorithms; debugging strategies; the concept and properties of algorithms

- Object-oriented principles (6 hours): abstraction, objects, classes, methods, parameter passing, encapsulation, inheritance, polymorphism
- Fundamental data structures (6 hours): primitive types, arrays, records, strings, references
- Secure code (2 hours): data encapsulation; information hiding and integrity; strict data typing
- Overview of operating systems (1 hour): role and purpose of operating systems; simple file management
- Human-computer interaction (1 hours): sound design concepts and fundamental graphical interface design; standard API graphics
- Program development (3 hours): program development phases, with emphasis on design, implementation, and testing and debugging strategies

Course Student Learning Outcomes:

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

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

CS I: Assessment Rubric for Student Learning Outcomes

Program Outcome	Student Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
2a,3b, 3c, 3d	Choose professional behavior in response to ethical issues inherent in computing.	**Explains** the concepts of intellectual property, plagiarism, and software piracy.	**Chooses** to respond professionally to ethical issues in computing, such as intellectual property, plagiarism, and software piracy.	**Values** and respects intellectual property, and **chooses** to act professionally.

1a, 1c	Produce algorithms for solving simple problems and trace the execution of computer programs.	**Defines** the steps necessary to solve a programming problem.	**Produces** a working programming solution for a given algorithm.	**Develops** a generic solution for an algorithm that can be used to solve a range of related problems.
1a, 2a	Compare and contrast the primitive data types of a programming language; describe how each is stored in memory; and identify the criteria for selection.	**Names** the built-in data types of the programming language.	**Differentiates** among the built-in data types and explain when it is appropriate to choose one over another.	Consistently **produces** programming solutions with the correct data types implemented.
1a, 1b, 1c, 1d	Apply the program development process to problems that are solved using fundamental programming constructs and predefined data structures.	**Summarizes** the phases of the program development cycle.	With guidance during the design phase, **produces** working code and performs some testing.	Develops a working program solution by implementing design, coding, and testing that includes error checking.
1a, 1b, 1c, 1d	Apply secure coding techniques to object-oriented programming solutions.	**Describes** secure coding techniques of an object-oriented program, such as public versus private members, data integrity, and data typing.	**Applies** secure coding techniques to an object-oriented program.	**Devises** a fully secure object-oriented program.

1a, 1c, 1d	Decompose a program into subtasks and use parameter passing to exchange information between the subparts.	With guidance **translates** a problem into a programming solution with subtasks.	With guidance for program analysis and design, **decomposes** a problem into program components that share data.	Independently **analyzes** a problem, **formulates** a design strategy, and **decomposes** a problem into program components that share data.
1a, 2c, 3a	Differentiate between the object-oriented, structured, and functional programming methodologies.	**Recognizes** the differences and similarities of the object-oriented, structured, and functional programming methodologies.	**Differentiates** between the object-oriented, structured, and functional programming methodologies.	**Compares** and **contrasts** the three prominent methodologies of object-oriented, structured, and functional programming.
1a, 2a, 3a	Describe the language translation phases of compiling, interpreting, linking and executing, and differentiate the error conditions associated with each phase.	**Defines** the programming language terms of compiling, interpreting, linking, executing, and error conditions.	**Describes** the programming language translation phases of compiling, interpreting, linking, and executing, and explains the types of errors associated with each phase.	**Compares** the programming language translation phases of compiling, interpreting, linking, and executing, and distinguishes the error conditions associated with each.

Course Title: Computer Science II

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

Course Prerequisite(s): Computer Science I

Course Co-requisite(s): Discrete Structures

Course Type: Computer Course

Eligible for Computing for Other Disciplines? Yes

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

Course Topics:

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

Course Student Learning Outcomes:

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

- Discuss significant trends and societal impacts related to computing, software and the Internet
- Construct object oriented programming solutions for reuse, using ADTs that incorporate encapsulation, data abstraction, and information hiding.
- Construct multiple-file or multiple-module programming solutions that use class hierarchies, inheritance, and polymorphism to reuse existing design and code.
- Design and develop secure and fault-tolerant programs that mitigate potential security vulnerabilities.
- Verify program correctness through the development of sound test plans and the implementation of comprehensive test cases.
- Create programming solutions that use data structures and existing libraries.

- Produce graphical user interfaces that incorporate simple color models and handle events.
- Analyze the execution of searching and sorting algorithms.

CS II: Assessment Rubric for Student Learning Outcomes

Program Outcome	Student Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
1a, 2a, 3b, 3c, 3d	Discuss significant trends and societal impacts related to computing, software, and the Internet.	**Explains** how databases and the Internet can impact privacy and property rights.	**Discusses** the potential uses and abuses of data and the consequences of the loss of privacy.	**Practices** ethical behavior when addressing property rights and privacy issues.
1a, 1b, 1c, 1d, 2a	Construct object oriented programming solutions for reuse, using ADTs that incorporate encapsulation, data abstraction, and information hiding.	**Summarizes** the concepts of encapsulation, data abstraction, and information hiding and explains how they apply to object-oriented programming.	**Organizes** programming solutions that include encapsulation, information hiding, and data abstraction.	**Constructs** reusable software components that incorporate encapsulation, data abstraction, and information hiding.
1a, 1b, 1c, 1d, 2a	Construct multiple-file or multiple-module programming solutions that use class hierarchies, inheritance, and polymorphism to reuse existing design and code.	**Describes** when inheritance and the use of class hierarchies is an appropriate design strategy	With guidance, **produces** a programming solution using inheritance and polymorphism	**Designs** and **constructs** a programming solution using the features of inheritance and polymorphism appropriately

1a, 1c, 1d, 2a, 3d	Design and develop secure and fault-tolerant programs that mitigate potential security vulnerabilities.	**Summarizes** important characteristics of software assurance, such as the elimination of buffer overflows, memory leaks and back-door access.	**Produces** a fault-tolerant program using the foundations of software assurance to mitigate potential security vulnerabilities.	**Designs** and **develops** a secure and fault-tolerant programming solution utilizing principles of software assurance.
1a, 1b, 2a, 2b	Verify program correctness through the development of sound test plans and the implementation of comprehensive test cases.	**Produces** test plans for object – oriented programming solutions that considers code coverage.	**Analyzes** a program and devises a test plan that examines code coverage and develops test cases for data coverage.	**Constructs** a test driver for code coverage and creates a formal test plan choosing comprehensive test cases for data coverage.
1a, 1b, 1c, 1d, 2a	Create programming solutions that use data structures and existing libraries.	**Produces** programming solutions that use existing library code.	**Organizes** programming solutions that incorporate appropriate data structures and pre-existing code.	**Designs** and **develops** programming solutions that use data structures, pre-existing libraries, and individual library code.
1a, 1d	Produce graphical user interfaces that incorporate simple color models and handle events.	**Differentiates** between good and bad design concepts for human-computer interfaces.	**Produces** programming code of a graphical user interface that utilizes a simple color model effectively and efficiently handles events triggered by user interaction.	**Develops** programming code for a graphical user interface that incorporates the concepts of good HCI design.

1a, 1b, 2a, 3a	Analyze the execution of searching and sorting algorithms.	**Describes** the execution trace of one searching algorithm and one sorting algorithm.	**Analyzes** the execution of various searching and sorting algorithms.	**Evaluates** the execution of various searching and sorting algorithms including a recursive solution.

Course Title: Computer Science III

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

Course Prerequisite(s): Computer Science II; Discrete Structures

Course Co-requisite(s): Calculus I

Course Type: Computer Course

Eligible for Computing for Other Disciplines? No

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

Course topics:

- Professionalism (1 hour): standards of professional behavior; professional computing societies and publications; professional responsibilities and liabilities; ACM Code of Conduct; career paths in computing
- Software engineering (4 hours): standard approaches and implementation tools for analysis and design; software lifecycle stages, processes and documentation; Software Process Maturity Scale
- Basic algorithmic analysis (3 hours): asymptotic analysis of upper and average complexity bounds; best, average, and worst case behaviors; big O and little o notations; standard complexity classes; empirical measurements of performance; time and space tradeoffs; recurrence relations
- Algorithmic strategies (2 hours): brute-force; greedy; branch-and-bound; heuristics; pattern matching; string/text
- Recursion (7 hours): recursive mathematical functions; divide-and-conquer, first-and-rest, and last-and-rest strategies; backtracking; recursion with linked lists, trees and graphs
- Canonical data structures (7 hours): stacks; queues; linked lists; hash tables; trees; graphs
- Formal computing algorithms (8 hours): efficiency of various sorting and searching algorithms; hashing; collision-avoidance strategies; binary search trees; depth- and breadth-first traversals; shortest-path algorithms; minimum spanning tree; transitive closure; topological sort

- Concurrency (2 hours): threads; scheduling, synchronization and timing; multi-threaded programs
- Software reuse (3 hours): design patterns; parametric polymorphism (templates or generics); code libraries; container classes and iterators
- Human-Computer Interaction (2 hours): universal principles; human-centered considerations; usability testing and verification; design trade-offs; secure user interfaces
- Software and Information Assurance (3 hours): conformance, trustworthiness, and predictable execution testing; exception handling; engineering and security trade-offs; risks and liabilities of computer-based systems; fault prevention in software lifecycle stages; intentional and unintentional software security vulnerabilities

Course Student Learning Outcomes:

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

- Practice the tenets of ethical and professional behavior promoted by professional societies; accept the professional responsibilities and liabilities associated with software development.
- Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software application that is fully implemented and formally tested.
- Compare and contrast a range of searching and sorting algorithms and analyze time and space efficiencies.
- Assess the appropriateness of using recursion to solve a given problem.
- Design and construct programming solutions using a variety of recursive techniques.
- Analyze the efficiency of recursive algorithms.
- Design and develop reusable software using appropriate data structures and templates.
- Create effective, efficient and secure software, reflecting standard principles of software engineering and software assurance.

CS III: Assessment Rubric for Student Learning Outcomes

Program Outcome	Student Learning Outcome	Approaches Goal	Meets Goal	Exceeds Goal
2a, 2b, 3a, 3b, 3c, 3d	Practice the tenets of ethics and professional behavior promoted by computing societies; accept the professional responsibilities and liabilities associated with software development.	***Studies*** the tenets of ethics and professional behavior promoted by international computing societies, such as ACM and IEEE-CS.	***Practices*** the tenets of ethics and professional behavior promoted by international computing societies, and recognizes the liabilities associated with software development.	***Displays*** ethical and professional behavior associated with the responsibilities of software development.
1a, 1b, 1c, 1d, 2a, 2b	Use standard analysis and design techniques to produce a team-developed, medium-sized, secure software application that is fully implemented and formally tested.	As part of a team, ***produces*** an executable, medium-sized software application that meets some program requirements and includes design documentation and some evidence of testing.	As part of a team, ***produces*** a working, medium-sized software application on time that meets many program requirements including design and some test plan documentation.	As part of a team, successfully ***develops*** a medium-sized, secure software application on time that meets all program requirements including design and formal test plan documentation.

1a, 1b, 2a	Compare and contrast a range of searching and sorting algorithms and analyze time and space efficiencies.	***Uses*** various searching and sorting algorithms, and investigates time and space tradeoffs.	***Compares*** and ***contrasts*** a range of searching and sorting algorithms for time and space efficiencies.	***Critiques*** searching and sorting algorithms, including recursive solutions, for various algorithmic efficiencies and evaluates them in terms of Big-O.
1a, 1b, 1c, 2a	Assess the appropriateness of using recursion to solve a given problem.	***Explains*** the utility of recursion to solve certain problems.	***Compares*** and contrasts the tradeoffs in terms of recursive and non-recursive solutions.	***Justifies*** when to choose a recursive solution over a non-recursive solution (and vice versa) in terms of efficiency, Big-O, and comprehensibility.
1a, 1d	Design and construct programming solutions using a variety of recursive techniques.	***Converts*** a simple recursive algorithm into a working recursive method.	With guidance ***develops*** recursive programming solutions for applications that use data structures such as trees and lists.	Independently ***designs*** and ***develops*** recursive programming solutions for applications that use backtracking and data structures such as trees and lists.
1a, 1b, 1c, 2a	Analyze the efficiency of recursive algorithms.	With guidance ***interprets*** a recursive method.	***Analyzes*** a recursive method and correctly predicts its output.	***Evaluates*** recursive algorithms in terms of efficiency and time and space tradeoffs.

1a, 1b, 1c, 1d, 2a, 2b	Design and develop reusable software using appropriate data structures and templates.	***Differentiates*** among the classic data structures and selects a suitable data structure for use in an application	With some guidance ***designs*** and ***develops*** applications using appropriate data structures for a given problem.	Independently ***designs*** and ***develops*** applications using appropriate data structures and incorporates reusable software components in the solution.
1a, 1b, 1c, 1d, 2a, 2b, 3a, 3b, 3c, 3d	Create effective, efficient and secure software, reflecting standard principles of software engineering and software assurance.	***Calculates*** the risks and liabilities of a computer-based solution using standard software assurance and engineering principles.	***Creates*** an effective, efficient and secure solution, utilizing principles of software assurance and software engineering.	***Judges*** the safety and security of a software solution.

Course Title: Algorithm Analysis and Design

Course Description: This course introduces formal techniques to support the analysis and design of algorithms; focusing on both the underlying mathematical theory and practical considerations of efficiency. Topics include: basic algorithmic analysis; fundamental algorithmic strategies; graph and tree algorithms; automata theory; and introduction to language translation.

Course Prerequisite: CS III

Course Co-requisite: none

Course Minimum Contact Hours: 42

Course Type: Computer Course

Eligible for Computing for Other Disciplines? No

Course Title: Computer Organization and Architecture

Course Description: This course covers basic hardware and software structure; I/O and main memory organization; internal representation of data; addressing methods; program controls; microprocessor families; multiprocessors; concurrent programming and synchronization; and RISC architectures.

Course Prerequisite: CS II

Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Hardware Fundamentals
Course Description: This course covers computer hardware components; I/O and communication interfaces; maintenance and troubleshooting; component swapping; system commands and utilities; memory managers; multi-core processors; and graphical user interface software.
Course Prerequisite: none
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Human-Computer Interaction
Course Description: This course covers the universal principles of interfaces between people and technology; usability testing and verification; human-centered software development and design trade-offs; design and programming of secure user interfaces; and gaming and multimedia systems.
Course Prerequisite: CS II
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Essentials of Computer Security
Course Description: This course covers the foundations of computing security: confidentiality, integrity, availability, authentication and non-repudiation. Topics also include business continuity and disaster recovery; ethics; digital forensics; security standards, regulatory compliance and organizational policies; and risk management.
Course Prerequisite: none
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Introduction to Database Systems
Course Description: This course covers information models and systems; database query languages; object-oriented and relational database design; transaction processing; distributed databases; data modeling; normalization; and physical database design.
Course Prerequisite: CS I
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Introduction to Software Engineering
Course Description: This course covers the basic principles and concepts of software engineering; system requirements; secure programming in the large; modeling and testing; object oriented analysis and design using the UML; design patterns; frameworks and APIs; client-server architecture; user interface technology; and the analysis design and programming of simple servers and clients.
Course Prerequisite: CS II
Course Co-requisite: CS III
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? No

Course Title: Linux Operating Environment
Course Description: This course introduces the concepts and features of the Linux operating system. Topics include file management; application installation; scripting; system and network configuration; kernel management; and an introduction to the LAMP (Linux, Apache, MySQL, Perl/PHP/Python) web server infrastructure.
Course Prerequisite: CS II
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: Programming Languages
Course Description: This course covers the fundamental concepts on which programming languages are formulated and the execution models that support them. Topics include: interpreting and compiling, formal descriptions of syntax and semantics; comparison of programming paradigms; data types, type checking, scope and binding; inheritance; distributed processing issues including exceptions and concurrency; and network programming.
Course Prerequisite: CS III
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? No

Course Title: Survey of Computing Disciplines
Course Description: This course is designed for students who intend to pursue a program of study in computing. Topics include an overview of the computing disciplines: Computer Science, Computer Engineering, Software Engineering, Information Systems, and Information Technology; the technical and professional preparation required for a variety of computing careers; workplace experiences; licensing and certification; as well as current and future trends in computing.
Course Prerequisite: none
Course Co-requisite: none
Course Minimum Contact Hours: 14
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

Course Title: XML-Enabled Technologies
Course Description: This course covers core XML-enabled applications and technologies for data exchange on the web. Topics include: creating well-formed XML documents; using namespaces; validating XML with DTDs and schemas; manipulating data using DOM and SAX; CSS and transforming XML documents with XSLT using XPATH; AJAX applications; XML databases; and Web Services using SOAP.
Course Prerequisite: CS II
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Computer Course
Eligible for Computing for Other Disciplines? Yes

SECTION 3.2: ENGINEERING COURSES

Course Title: Digital Logic Circuits with Lab
Course Description: This course covers the foundations of microprocessor design and execution. Topics include number systems and two's-complement arithmetic; Boolean algebra; logic design; gates; flip-flops; registers; sequential circuits; control mechanisms; timing; data and control flow in a typical computer. These topics are supported and reinforced with hands-on laboratory activity designed to enhance the understanding and proper use of selected principles from digital logic circuit theory.
Course Prerequisite: CS I and Discrete Structures
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Engineering Course

Course Title: Circuit Analysis with Lab
Course Description: This course covers DC resistive circuits; Kirchhoff's Laws; Nodal and Mesh emphasis; sources; Thevenin's and Norton's theorems; RC, RL and RCL circuit solutions; and sinusoidal steady state solutions. These topics are supported and reinforced with hands-on laboratory activity designed to enhance the understanding and proper use of selected principles from circuit analysis.
Course Prerequisite: Calculus II; Physics I
Course Co-requisite: none
Course Minimum Contact Hours: 42
Course Type: Engineering Course

SECTION 3.3: MATHEMATICS COURSES

Course Title: Calculus I
Course Description: This course is the first in the calculus sequence designed for the engineering, science, or mathematics major. Topics include functions and limits, techniques and applications of differentiation, indefinite and definite integrals, and applications of integration.
Course Prerequisite: Precalculus with Trigonometry
Course Co-requisite: none
Course Minimum hours: 56
Course Type: Mathematics Course

Course Title: Discrete Structures
Course Description: The course covers mathematical topics essential for work in computer science. Topics include: number bases, mathematical induction, sets, relations, functions, congruence, recursion, combinations and permutations, probability, graphs, trees, logic, Boolean algebra, and proof techniques. Computing related problems and examples are integrated throughout the course.
Course Prerequisite(s): Pre-calculus
Course Minimum Contact Hours: 42 (hours per course topic are suggested below)
Course Type: Mathematics Course
Course Topics:

- **Number bases (1 hour)**: binary, hexadecimal.
- **Mathematical induction (4 hours)**: examples of mathematical induction; strong induction.
- **Sets, relations functions, congruences (9 hours)**: sets including Venn diagrams, complements, power sets, operations, DeMorgan's laws; relations including equivalence relations, equivalence classes; functions including injective, surjective, inverse, composition, domain, co-domain, range.
- **Recursion (4 hours)**: recursive definitions of functions; factorials; Fibonacci sequences; Towers of Hanoi; other functions and sequences.
- **Combinatorics (7 hours)**: binomials; counting arguments; discrete probability; combinations and permutations; pigeon-hole principle.
- **Graphs and trees (11 hours)**: directed graphs; undirected graphs; weighted graphs; Eulerian and Hamiltonian circuits; traveling sales person; graph coloring; trees (binary, spanning); expression trees; tree traversals.
- **Logic and Boolean algebra (3 hours)**: truth tables; propositional calculus; Boolean algebra and Boolean circuits.
- **Other proof techniques (3 hours)**: direct proof; proofs by counter example, contrapositive, and contradiction; logical equivalence and circles of implication.

Course Student Learning Outcomes:

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

- Perform binary and hexadecimal number conversions.
- Apply mathematical induction and other techniques to prove mathematical results.
- Solve problems involving sets, relations, functions, and congruencies.
- Perform computations using recursively defined functions and structures.
- Use methods of combinatorics to solve counting problems.
- Illustrate the basic terminology and properties of graphs and trees.
- Use graphs and trees to solve problems algorithmically.
- Examine the logical validity of arguments and proofs as they apply to Boolean expressions

Discrete Structures: Assessment Rubric for Student Learning Outcomes

Program Outcome	Student Learning Outcome	Approaches Goal	Meets Goal	Surpasses Goal
1a	Perform binary and hexadecimal conversions of numbers	***Converts*** binary numbers to their decimal equivalent	***Converts*** positive numbers between bases 2, 10, and 16	***Performs*** two's complement to represent negative integers in binary
1a, 1b, 1c, 2a	Apply mathematical induction and other techniques to prove mathematical results	***Recognizes*** valid proofs that use mathematical induction and other techniques	Given a simple problem, such as an identity, ***constructs*** a mathematical proof by induction	***Constructs*** mathematical proofs by induction and other techniques
1a, 1b, 1c	Solve problems involving sets, relations, functions, and congruences	***Defines*** the concepts of sets, relations, functions, and congruences	***Solves*** problems about sets, relations, functions, and congruences	***Evaluates*** a problem and constructs an appropriate solution choosing among sets, relations, functions, and/or congruences
1a, 1b, 1c	Perform computations using recursively defined functions and structures	***Explains*** how a simple recursive function is evaluated	***Computes*** the correct result produced by a recursive algorithm	***Constructs*** recursive algorithms for the solution of problems

1a, 1b, 1c, 2a	Use methods of combinatorics to solve counting problems	***Recognizes*** the need for combinatorial techniques such as combinations or permutations to solve a problem	***Solves*** counting problems using combinatorial techniques such as combinations and permutations	***Decomposes*** a complex problem into combinatorial procedures
1a, 1b, 1c, 2a	Illustrate the basic terminology and properties of graphs and trees	***Defines*** terms and properties for graphs and trees	Given a problem description ***illustrates*** appropriate trees, binary search trees, weighted, directed and undirected graphs solutions	***Applies*** mathematical proofs to verify the properties of graphs
1a, 1b, 1c, 2a	Use graphs and trees to solve problems algorithmically	***Explains*** standard algorithms for graphs and trees, such as Eulerian circuits, spanning trees, and Kruskal's algorithm	***Applies*** traversal methods for graphs and trees	***Verifies*** the correctness of graph algorithms using mathematical proofs
1a, 1b, 1c, 2a	Examine the logical validity of arguments and proofs as they apply to Boolean expressions	***Identifies*** the properties and structures of Boolean algebra	***Analyzes*** the steps to simplify a Boolean expression	***Constructs*** a proof using the laws of Boolean algebra

APPENDIX A: CUSTOMIZED BLOOM'S TAXONOMY

The *Taxonomy of Educational Objectives*, often called Bloom's Taxonomy, is a classification of the different learning outcomes that educators set for students. Bloom's Taxonomy divides educational objectives into three domains: Affective, Psychomotor, and Cognitive. In Bloom's hierarchical taxonomy, achievement at the higher levels is dependent on having attained prerequisite knowledge at lower levels.

The ACM Two-Year College Education Committee has adapted Bloom's original taxonomy in developing a rubric for the assessment of student learning outcomes in its computing curricula. This customization includes only the relevant domains, Cognitive and Affective. The levels identified in the Cognitive domain revolve around knowledge, comprehension, and thinking through a particular topic. In its computing curricula, the ACM Two-Year College Education Committee uses the Cognitive domain to assess student mastery of technical subject matter. There are six levels in the taxonomy for the Cognitive domain, progressing from the lowest order processes to the highest:

1. **Knowledge -** The ability to recall what has been learned.
2. **Comprehension -** The ability to demonstrate a basic understanding of communicated material or information.
3. **Application -** The ability to put basic rules, conventions, ideas and concepts together to solve new problems.
4. **Analysis -** The ability to deconstruct information logically into components to ascertain interrelationships and to distinguish between facts and inferences.
5. **Synthesis -** The ability to assemble ideas creatively to design or develop a new or unique product or structure.
6. **Evaluation -** The ability to judge the value or usefulness of materials, ideas or information based on established standards and criteria.

The levels identified in the Affective domain describe the way people react emotionally and their ability to feel empathy for another. Affective outcomes typically assess awareness and growth in attitudes, emotion, and feelings. In its computing curricula, the ACM Two-Year College Education Committee uses the Affective domain to assess student demonstration of soft skills such as professional behavior, ethical conduct, and social awareness of the impact of technology. There are five levels in the affective domain progressing from the lowest order processes to the highest:

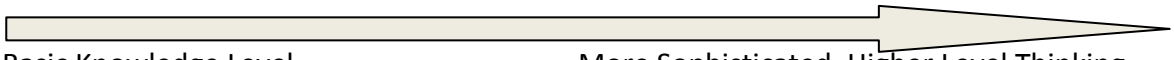
1. **Receiving** – Students become aware of an attitude, behavior, or value.
2. **Responding** – Students exhibit a reaction or change as a result of exposure to an attitude, behavior, or value.
3. **Valuing** – Students recognize value and display this through involvement or commitment.

4. **Organizing** – Students determine a new value or behavior as important or a priority.
5. **Characterizing** – Students integrate consistent behavior as a naturalized value in spite of discomfort or cost. The value is recognized as a part of the person's character.

Cognitive Domain

Student Learning Outcomes related to knowledge

The tasks charted below increase in sophistication moving from left to right.

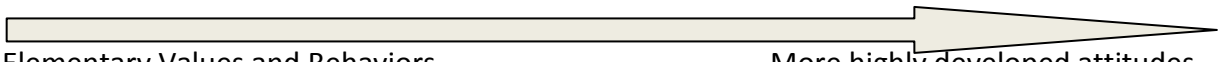


Basic Knowledge Level — More Sophisticated, Higher Level Thinking, Critical Thinking

Knowledge	**Comprehension**	**Application**	**Analysis**	**Synthesis**	**Evaluation**
Define	Convert	Apply	Analyze	Compose	Appraise
Identify	Demonstrate	Calculate	Categorize	Construct	Assess
Label	Describe	Diagram	Compare	Create	Choose
List	Differentiate	Edit	Contrast	Design	Critique
Name	Discuss	Illustrate	Decompose	Develop	Debate
Recall	Explain	Investigate	Deduce	Hypothesize	Defend
Recognize	Interpret	Manipulate	Devise	Invent	Estimate
Select	Paraphrase	Modify	Dissect	Reconstruct	Evaluate
Show	Summarize	Produce	Distinguish	Reorganize	Judge
State	Translate	Relate	Examine	Schematize	Justify
Visualize		Solve	Organize		Recommend
		Transform	Plan		Verify
		Use			

Affective Domain

Student Learning Outcomes related to attitudes, behaviors, and values



Elementary Values and Behaviors — More highly developed attitudes

Receiving	**Responding**	**Valuing**	**Organizing**	**Characterizing**
Attend	Behave	Accept	Adapt	Authenticate
Describe	Comply	Balance	Adjust	Characterize
Explain	Cooperate	Choose	Alter	Defend
Locate	Discuss	Differentiate	Change	Display
Observe	Examine	Influence	Develop	Embody
Realize	Follow	Prefer	Improve	Habituate
Receive	Model	Seek	Modify	Internalize
Recognize	Present	Value	Practice	Produce
	Respond		Revise	Represent
	Show			Validate
	Study			Verify

APPENDIX B: REFERENCES

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

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

IEEE-CS/ACM Joint Curriculum Task Force*, Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.*

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

Association for Computing Machinery, Inc. & the Institute for Electrical and Electronics Engineers, Inc. *Software Engineering Code of Ethics and Professional Practice.* (1999). Retrieved July 12, 2005 from http://www.acm.org/serving/se/code.htm.

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

IEEE-CS/ACM CC2001 Task Force, *Computing Curricula 2001 Final Draft - December 15, 2001,* http://www.computer.org/education/cc2001/final/index.htm

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

IEEE-CS/ACM Joint Curriculum Task Force, *Computing Curricula 2001: Computer Science*, http://acm.org/education/curric_vols/cc2001.pdf

IEEE-CS/ACM Joint Curriculum Task Force*, Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering,* http://www.acm.org/education/CE-Final-Report.pdf

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

National Council of Engineering Examiners, *NCEE Model Rules of Professional Conduct*, http://www.ncees.org/

ABET, Inc., http://www.abet.org/

ACM *Code of Ethics and Professional Conduct*, http://www.acm.org/constitution/code.html

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