Computing Curricula 2007: Guidelines for Associate-Degree Transfer Curriculum in Computer Engineering

www.acmtyc.org

The ACM Two-Year College Education Committee

Robert D. Campbell, CUNY Graduate Center
Committee Chair

Elizabeth K. Hawthorne, Union County College

Karl J. Klee, Alfred State College

This material is based in part on work supported by the National Science Foundation under Grant Number 0229748.

Computing Curricula 2007: Guidelines for Associate-Degree Transfer Curriculum in Computer Engineering

The ACM Two-Year College Education Committee
and
The Joint Task Force on Computer Engineering

Association for Computing Machinery
IEEE Computer Society

www.acm.org
www.acmtyc.org
www.computer.org

Acknowledgements

The ACM Two-Year College Education Committee gratefully acknowledges the outstanding contributions to the development of this report provided by our colleagues, and especially John Impagliazzo, as well as the previous work by the joint ACM/IEEE-CS Computer Engineering Task Force in the development of the baccalaureate report *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*.

The ACM Two-Year College Education Committee expresses its gratitude to the following colleagues for their thoughtful reviews and recommendations in the development of this report.

George Cheng
Hostos Community College, NY

Peter Drexel
Plymouth State University, NH

Becky Grasser
Lakeland Community College, OH

Haibo He
Stevens Institute of Technology, NJ

John Impagliazzo
Hofstra University, NY

Victor Nelson
Auburn University, AL

Stacie Lynch Newberg
Western Wyoming Community College, WY

Pradip Srimani
Clemson University, SC

Michael Wolf
New Mexico State University, NM

Anita Wright
Camden County College, NJ

Table of Contents

List of Tables

(This page intentionally left blank.)

Section 1: The Goal of This Report

This report provides guidelines for a computer engineering transfer degree program at associate-degree granting institutions. These guidelines present a program of study designed for students intending to transfer into baccalaureate programs awarding computer engineering degrees. Specifically designed to facilitate student transfer by linking computer engineering curricula in two-year colleges with those in baccalaureate institutions, this report promotes articulation and student success.

There are two major curriculum reports that provide foundation for this work:

- Computer Engineering curricula guidelines for undergraduate baccalaureate-degree programs were finalized and approved in 2004, published under the title *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* This work was the result of a joint task force of the ACM and IEEE-CS. That report, together with accompanying materials, can be found at http://www.computer.org/education/.

- Computer Science curricula guidelines for associate-degree granting institutions were finalized and approved in 2003, published under the title *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science.* This work was the result of the IEEE-CS/ACM Joint Task Force on Computing Curricula 2001 and the ACM Two-Year College Education Committee. That report, together with accompanying materials, can be found at http://www.acmtyc.org/.

This report, *Guidelines for Associate-Degree Transfer Curriculum in Computer Engineering*, shares common goals and outcomes with the two above-mentioned curriculum reports. According to the American Association for Community Colleges, nearly one-half of all undergraduates in the United States are enrolled in community colleges. For this reason, it is important to outline a computer engineering degree program that can be initiated in the two-year college setting, and is designed for seamless transfer into an upper division program. This report recommends a program of study that specifically fulfills that requirement. However, it must be noted that the aims and objectives for computer engineering undergraduate degree programs can vary from one institution to another for a variety of reasons. Ultimately, students are best served when institutions establish well defined articulation agreements between associate-degree and baccalaureate-degree programs.

It is critical to note that two-year college students must complete the associate-level coursework in its entirety to well-defined competency points to ensure success in the subsequent computer engineering coursework at the upper division level. For many students, this may require more than two years of study at the associate level. Particular attention must be paid to matching individual students to appropriate programs of study, taking into account each student's career goals and aspirations, talents and abilities, and life constraints such as time, finances, and geography. Students seeking a path to the workforce at the conclusion of an associate's degree will not find these *Guidelines* to be directly applicable.

Some students initiate their college studies uncertain that college is right for them, with unclear goals or undetermined career plans; they may not yet know whether the discipline they have initially chosen is a proper or wise selection for them. Students of this nature who are attracted to computer engineering might well be served by a program that is intended to produce computer engineering technicians; such programs typically use internships to give students valuable opportunities to work with professionals practicing in the discipline. Students who go on to become technicians gain on-the-job insights into future career paths in the field and may subsequently decide that they want to advance beyond the technician level. This will require completion of a degree program of the nature described in these *Guidelines*, followed by transfer and completion of a baccalaureate degree.

By basing this report on two published sets of curricula guidelines, the following goals are fulfilled:

- The incorporation of the computer engineering philosophy, concepts, coursework and outcomes from the baccalaureate *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report helps to prepare students properly and facilitates seamless articulation. For those familiar with the baccalaureate-degree guidelines, the goal is to prepare students for transfer into any of Curriculum Implementations A, B or C, depending on specific course selections.
- The use of model courses from the *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* report enables two-year colleges in the United States to implement a computer engineering degree program using an established core of computer science, mathematics and science courses.

Section 2: The Nature of Computer Engineering

The baccalaureate *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report provides insights into the nature of this field.

> *Computer engineering is defined as the discipline that embodies the science and technology of design, construction, implementation, and maintenance of software and hardware components of modern computing systems and computer-controlled equipment. Computer engineering has traditionally been viewed as a combination of both computer science (CS) and electrical engineering (EE). It has evolved over the past three decades as a separate, although intimately related, discipline. Computer engineering is solidly grounded in the theories and principles of computing, mathematics, science, and engineering and it applies these theories and principles to solve technical problems through the design of computing hardware, software, networks, and processes.*
>
> *Historically, the field of computer engineering has been widely viewed as "designing computers." In reality, the design of computers themselves has been the province of relatively few highly skilled engineers whose goal was to push forward the limits of computer and microelectronics technology. The successful miniaturization of silicon devices and their increased reliability as system building blocks has created an environment in which computers have replaced the more conventional electronic devices. These applications manifest themselves in the proliferation of mobile telephones, personal digital assistants, location-aware devices, digital cameras, and similar products. It also reveals itself in the myriad of applications involving embedded systems, namely those computing systems that appear in applications such as automobiles, large-scale electronic devices, and major appliances.*
>
> *Increasingly, computer engineers are involved in the design of computer-based systems to address highly specialized and specific application needs. Computer engineers work in most industries, including the computer, aerospace, telecommunications, power production, manufacturing, defense, and electronics industries. They design high-tech devices ranging from tiny microelectronic integrated-circuit chips, to powerful systems that utilize those chips and efficient telecommunication systems that interconnect those systems. Applications include consumer electronics (CD and DVD players, televisions, stereos, microwaves, gaming devices) and advanced microprocessors, peripheral equipment, systems for portable, desktop and client/server computing, and communications devices (cellular phones, pagers, personal digital assistants). It also includes distributed computing environments (local and wide area networks, wireless networks, internets, intranets), and embedded computer systems (such as aircraft, spacecraft, and automobile control systems in which computers are embedded to perform various functions). A wide array of complex technological systems, such as power generation and distribution systems and modern processing and manufacturing plants, rely on computer systems developed and designed by computer engineers.*
>
> *Technological advances and innovation continue to drive computer engineering. There is now a convergence of several established technologies (such as television, computer, and networking technologies) resulting in widespread and ready access to information on an enormous scale. This has created many opportunities and challenges for computer engineers. This convergence of technologies and the associated innovation lie at the heart of economic development and the future of many organizations. The situation bodes well for a successful career in computer engineering.*

The baccalaureate *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report also provides useful distinctions among various computer-related career options.

> *Electrical engineering spans a wide range of areas, including bioengineering, power engineering, electronics, telecommunications and digital systems. Related to the field of computer engineering, electrical engineers concern themselves primarily with the physical aspects of electronics including circuits, signal analysis, and microelectronic devices. Computer scientists concern themselves primarily with the theoretical and algorithmic aspects of computing with a focus on the theoretical underpinnings of computing. Software engineers have a focus on the principles underlying the development and maintenance of correct (often large-scale) software throughout its lifecycle. Information systems specialists encompass the acquisition, deployment, and management of information resources for use in organizational processes. Information technology specialists would focus on meeting the needs of users within an organizational and societal context through the selection, creation, application, integration, and administration of computing technologies. Computer engineering technologists support engineers by installing and operating computer-based products, and maintaining those products.*

Finally, the baccalaureate *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report discusses the important role that professionalism plays in the field of Computer Engineering.

> *The public has entrusted in engineers a level of responsibility because the systems they design (whether x-ray machines, air traffic control systems, or nuclear power plants) affect the public directly and indirectly. Therefore, it is incumbent upon computer engineers to exercise the utmost conscientiousness in their designs and implementations of computing systems. As such, graduates should have an understanding of the responsibilities associated with engineering practice, including the professional, societal, and ethical context in which they do their work. Such responsibilities often involve complicated trade-offs involving fiscal and social contexts. This social context encompasses a range of legal and economic issues such as intellectual property rights, security and privacy issues, liability, technological access, and global implications and uses of technologies. Professionalism and ethics are critical elements, since the focus of engineering on design and development makes social context paramount to studies in the field. Computer engineering students must learn to integrate theory, professional practice, and social constructs in their engineering careers. It is incumbent upon all computer engineers to uphold the tenets of their profession and to adhere to the codes of professional practice. The public expects engineers to follow prescribed rules of professional practice and to not engage in activities that would tarnish their image or that of their practicing colleagues.*

Section 3: Computer Engineering Curricular Considerations

Robust studies in mathematics and science are absolutely critical to student success in the pursuit of computer engineering. Mathematical and scientific concepts and skills must be understood and mastered in a manner that enables the student to draw on these disciplines throughout the computer engineering curriculum. One cannot overstate the role that mathematics and science play in underpinning an engineering student's academic pursuits.

A strong and extensive foundation in mathematics provides the necessary basis for studies in computer engineering. This foundation must include both mathematical techniques and formal mathematical reasoning. Mathematics provides a language for working with ideas relevant to computer engineering, 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 corequisite structures, and learning activities and laboratory assignments must be designed to reflect and support this framework. Specific mathematical content must include the principles and techniques of discrete structures; this concept is reflected in the extensive body of knowledge and sample courses outlined in this report. Furthermore, students must master the established sequence in differential and integral calculus, also included in the curriculum mapping identified in this report.

Rigorous laboratory science courses provide students with content knowledge as well as experience with the "scientific method," which can be summarized as formulating problem statements and hypothesizing; designing and conducting experiments; observing and collecting data; analyzing and reasoning; and evaluating and concluding. For students pursuing the field of computer engineering the scientific method provides a baseline methodology for much of the discipline; it also provides a process of abstraction that is vital to developing a framework for logical thought. Learning activities and laboratory assignments found in specific computer engineering courses should be designed to incorporate and reinforce this framework. Specific science coursework should include the discipline of physics, which provides the foundation and concepts that underlie the electrical engineering content reflected in the body of knowledge in this report. Additional natural science courses, such as chemistry and biology, can provide important content for distinct specializations within computer engineering; such considerations will vary by institution based on program design and resources.

Engineering courses in the lower division serve two important functions: first, to familiarize students with the engineering disciplines, and second to establish a strong foundation for advanced coursework in their chosen specialization. It is important to engage students' innate interests early in their academic careers to cement their commitment to engineering, to further student retention, and to motivate achievement in their coursework. The computer engineering courses in this curriculum model are foundational and standard, hence especially well-suited for transfer to a baccalaureate program of study.

Clearly a program in computer engineering requires a solid foundation in computer science, beyond mere introductory experiences. A robust lower division course of study in computer science – as defined in *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* – serves this requirement well. Furthermore, because the relationships among mathematics, computer science and engineering courses are inherent, topics in these disciplines can be interwoven; these intrinsic relationships should be nurtured as the program of study unfolds.

The engineering laboratory experience is another essential part of the computer engineering 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 engineering. Computer engineering 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 computer engineering students.

The following important points about engineering laboratories are derived from the *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report:

- Introductory laboratories should be designed and conducted to reinforce concepts presented in lecture classes and homework. Activities should demonstrate specific phenomena or behavior and provide experiences with measuring and studying desired characteristics.
- Laboratories should include the physical implementation of designs such as electronic and digital circuits, bread-boarding, microprocessor interfacing, prototyping, and implementation of hardware and software.
- Laboratories should include application and simulation software to design small digital and computer systems. The use of simulation tools to model and study real systems is often desirable and necessary to allow students to study systems that are not practical to design and implement physically.
- Students should learn to record laboratory activity to document design activities, experiments, and measured/observed results. Students should use these opportunities to further develop their technical writing abilities.
- Assignments in a laboratory setting should be practical and carefully planned to develop confidence in the technical abilities of students.
- The laboratory experience also should ensure that students:
    - Practice safety in all laboratories, especially where electronic equipment and electricity pose dangers.
    - Use microprocessors and test equipment appropriately.
    - Build electronic circuits and devices to design specifications.
    - Summarize the processes and concerns associated with product development and manufacturing.
    - Recognize and resolve trade-offs between hardware and software.
    - Treat laboratories as places of serious study and endeavor.
    - Are captivated by the allure of computer engineering.

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. Likewise, many institutions currently conduct engineering-related courses or professional development activities in service to their career-track students or their local industry base. These colleges will find that they can leverage existing facilities, resources and faculty expertise in implementing a transfer program in computer engineering. However, lower division engineering courses should be taught by faculty with engineering credentials to ensure that the courses have credibility, reflect the real world practices of engineering, and properly prepare students for the upper division engineering curriculum.

In addition to the scientific and technical content noted above, effective abilities in oral and written communication are of critical importance to computer engineering professionals; these skills must be established, nurtured and incorporated throughout a computer engineering 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 benefit a computer engineering 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.

Professional, legal and ethical issues are important elements in the overall computer engineering curriculum, and must be integrated throughout the program 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. In addition, the *Model Rules Of Professional Conduct* issued by the National Council of Engineering Examiners (NCEE) include the tenets that practitioners "shall be objective and truthful in professional reports, statements or testimony" and shall "hold paramount the safety, health and welfare of the public in the performance of their professional duties." Again, these ethics should be incorporated into instructional activities wherever possible.

Colleges must ensure that degree programs ultimately fulfill all general education and related requirements arising from institutional, state, and regional accreditation guidelines. The curriculum recommendations contained herein are intended to be compatible with those requirements, but recognize that in some instances, institutions may find it necessary to make specific alterations.

Articulation agreements often guide curriculum content as well, and are important considerations in the formulation of programs of study, especially for transfer-oriented programs. Institutions are encouraged to work collaboratively to design compatible and consistent programs of study that enable students to transfer from associate-degree programs into baccalaureate-degree programs.

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 computer engineers. 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.

Section 4: The Computer Engineering Transfer Curriculum Model

Table 1 provides a model for the sequencing of the courses constituting the transfer degree program in Computer Engineering. This program assumes that students enter with sufficient mathematics background to engage the study of the calculus; under-prepared students should be counseled to complete the necessary prerequisite courses before commencing the engineering program. The template below incorporates the appropriate prerequisite considerations and reflects the guidance found in Section 3 of this report.

Semester 1	**Semester 2**
Calculus I	Calculus II
Natural Science I	Natural Science II
Computer Science I	Computer Science II
Introduction to Engineering	Discrete Structures I
English I	English II
Semester 3	**Semester 4**
Calculus III	Mathematics, *select from:* • Discrete Structures II • Differential Equations • Linear Algebra • Probability & Statistics
Physics I	Physics II
Computer Science III	Engineering Course
Engineering Course	Engineering Course
General Education	General Education

Table 1: Computer Engineering Transfer Program

The engineering courses should be selected from Digital Logic Circuits, Computer Organization and Architecture, and Circuit Analysis; these courses – together with Introduction to Engineering – are specifically intended to support transfer into the upper division of baccalaureate programs in Computer Engineering. Course descriptions can be found in Appendix A. The engineering courses described in this report equate with corresponding courses in Curriculum Implementations A, B and C found in the *Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering* report. These course equivalences are cross-referenced in Table 2 and can be used to facilitate transfer and articulation agreements.

Guidelines for Associate-Degree Transfer Curriculum in Computer Engineering	*Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*		
	Implementation A	**Implementation B**	**Implementation C**
Introduction to Engineering	(None)	Engineering Problem Solving	Introduction to Engineering
Digital Logic Circuits with Lab	Digital Logic with Lab	Digital Logic Circuits with Lab	Digital and Logic Design with Lab
Computer Organization and Architecture	Computer Organization	Computer Organization	Introduction to Computer Organization
Circuit Analysis with Lab	Circuits and Systems with Lab	Electric Circuits with Lab	Engineering Circuit Analysis with Lab

Table 2: Engineering Course Equivalencies

The required mathematics courses identified in Table 1 include the standard course sequence in calculus offered for science and engineering majors, as well as the first course in discrete mathematics. (Engineering students exceed the stated "precalculus-ready" prerequisite for discrete mathematics.) Other courses available for selection include the second course in discrete mathematics, as well as the standard courses in ordinary differential equations, linear algebra, and probability and statistics commonly offered at the sophomore level for science and engineering majors. The descriptions for the courses in discrete mathematics can be found in Appendix B; for more comprehensive course details and learning outcomes consult the *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* report.

The computer science course sequence identified in Table 1 should be selected from either the imperative-first paradigm (CS101I -102I -103I) or the objects-first paradigm (CS101o-102o-103o), subject to articulation and transfer agreements. The descriptions for each of these three-course sequences can be found in Appendix C; for more comprehensive course details and learning outcomes consult the *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* report.

The designation *Natural Science I-II* in Table 1 should be selected from the standard lab-based course sequences for majors in chemistry, biology, or some other natural science appropriate to the student's course of study. The *Physics I-II* course sequence is the standard calculus-based physics (with lab) commonly offered for science and engineering majors in the lower division.

The general education courses identified in Table 1 should be selected to address institutional requirements and to complement the study of computer engineering. In addition to the English, mathematics, science, computing and engineering courses identified in Table 1, there are designations for the two general education course requirements typically allocated to the social sciences and humanities. Some institutions may have requirements that necessitate more or fewer such courses.

Bibliography

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

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

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/

University of Notre Dame, College of Engineering, http://www.cse.nd.edu/courses/eg111/www/course/course.htm

Appendix A
Engineering Course Descriptions

Introduction to Engineering

This course provides an overview of the engineering profession, its genesis and evolution to the present day, including fields of engineering and career paths within same; study of ethics with an emphasis on the engineering workplace; engineering design and analysis techniques, development of problem-solving skills, communication skills; student design projects.

Prerequisites: none

Syllabus and Student Performance Objectives:

Engineering as a profession

- Examine samples of engineering projects.
- Design, conduct, and analyze simple engineering projects.
- Examine what engineering entails and encompasses.
- Critique engineering as a career path.
- Define the major types of activities of different engineering disciplines.
- Define department specific curricula and opportunities.
- Recognize the interdisciplinary nature of engineering.

Engineering skills

- Identify, define and formulate an engineering problem to peak student interest.
- Apply simple analytical and empirical methods to solve a problem.
- Identify and utilize physical principles and laws appropriate to the problem.
- Apply scientific problem solving methods to engineering projects.
- Design processes on a simulated engineering project.
- Apply hardware and software usage in engineering projects.
- Identify positive and negative attributes of team dynamics.
- Produce interesting solutions to engineering problems in both individual and group settings.

Communications

- Compose written reports using technical writing skills.
- Describe a project orally using appropriate engineering terminology.
- Design visuals to support oral presentations.
- Demonstrate effective interpersonal skills working as a member of a team.

Design experience

- Produce a plan to meet design objectives.
- Design, build and demonstrate an artifact made to accomplish a task.

Human-Computer Interaction (HCI)

- Identify some contributors to human-computer interaction and relate their achievements to the knowledge area.
- Justify proper HCI designs in engineering.
- Compare and contrast small-screen and large screen graphical user interfaces
- Evaluate an engineering design using some principles of HCI.
- Describe how computer engineering uses or benefits from good HCI design.
- Create a conceptual vocabulary for analyzing human interaction with software to include terms such as affordance, conceptual model, and feedback.
- Summarize the basic science of psychological and social interaction relevant to the development of human-computer interfaces.
- Distinguish between a hypothesis and experimental results.
- Evaluate and analyze experimental results vis-à-vis hypothesis, recognizing the role of correlation.
- Distinguish between the different interpretations that a given icon, symbol, word, or color can have in different human cultures as well as in the context of human diversity.
- Create and conduct a simple usability test for an existing software application, taking into account human diversity.

Social and professional issues

- Identify some contributors to social and professional issues and relate their achievements to the knowledge area.
- Contrast between ethical and legal issues.
- Contrast between a patent and a copyright.
- Identify some ways in which a person can be credentialed to practice computer engineering.
- Describe issues that contrast risk issues with safety issues.
- Identify some issues in computer engineering that address privacy.
- Describe whistle blowing and discuss the conflicts between ethics and practice that may result from doing so.
- Describe how computer engineering uses or benefits from social and professional issues.
- Interpret the social context of a particular implementation.
- Identify assumptions and values embedded in a particular design.
- Evaluate a particular implementation using empirical data.
- Identify positive and negative ways in which computing alters the modes of interaction between people.
- Explain why computing/network access is restricted in some countries.

Digital Logic Circuits with Lab

This course includes the topics of internal structures of computers; number systems and arithmetic, two's-complement arithmetic; Boolean algebra, logic design, gates, synthesis of combinatorial networks; 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. Experiments introduce basic techniques and problem-solving, while comparisons between theoretical and experimental results are investigated in written laboratory reports.

Prerequisite: Computer Science I
Pre- or Corequisite: Discrete Structures I

Syllabus and Student Performance Objectives:

History and overview

- Discuss the achievements of key contributors to the field of digital logic.
- Explain why Boolean logic is important to this subject.
- Describe why gates are the fundamental elements of a digital system.
- Contrast the difference between a memory element and a register.
- List some uses for sequential logic.
- Summarize how the concepts of digital systems/logic apply to computer engineering.

Switching theory

- Perform arithmetic using the binary number system.
- Demonstrate the switching functions that underpin digital circuits.
- Analyze switching functions and their associated simplified circuits.

Combinational logic circuits

- Interpret switching functions as networks of logic gates.
- Explain and apply fundamental characteristics of relevant electronic technologies, such as propagation delay, fan-in, fan-out, power dissipation and noise margin.

Modular design of combinational circuits

- Analyze and explain uses of small- and medium-scale logic functions as building blocks.
- Analyze and design combinational logic networks in a hierarchical, modular approach, using standard and custom logic functions.

Memory elements

- Design and describe the operation of basic memory elements.
- Analyze circuits containing basic memory elements.
- Apply the concepts of basic timing issues, including clocking, timing constraints, and propagation delays during the design process.

Sequential logic circuits

- Analyze the behavior of synchronous and asynchronous machines.
- Create synchronous and asynchronous sequential machines.

Digital systems design

- Apply digital system design principles and descriptive techniques.
- Analyze and design functional building blocks and control and timing concepts of digital systems.
- Produce a complex digital system design in a hierarchical fashion using top-down and bottom-up design approaches.
- Justify programmable devices such as FPGAs and PLDs to implement digital system designs.
- Apply Hardware Description Language (HDL) for digital logic design.

Computer Organization and Architecture with Lab

This course includes the topics of basic hardware and software structure, addressing methods, programs control, processing units, I/O organization, arithmetic, main-memory organization, peripherals, microprocessor families, RISC architectures, and multiprocessors. These topics are supported and reinforced with hands-on laboratory activity designed to enhance the understanding and proper use of selected principles from computer organization and architecture. Experiments introduce basic techniques and problem-solving, while comparisons between theoretical and experimental results are investigated in written laboratory reports.

Pre- or Corequisite: Computer Science III

Syllabus and Student Performance Objectives:

History and overview of computer organization and architecture

- Discuss the achievements of key contributors to the field of computer architecture and organization.
- List the reasons and describe the strategies for different architectures.
- Identify differences between computer organization and computer architecture.
- Describe the major components of a computer.
- Analyze strengths and weaknesses inherent in different architectures.
- Summarize how the concepts of computer architecture and organization apply to computer engineering.

Fundamentals of computer architecture

- Describe the organization of a von Neumann machine and its major functional units.
- Explain how a computer fetches from memory and executes an instruction.
- Compare and contrast the strengths and weaknesses of the von Neumann architecture.
- Explain the relationship between the representation of machine level operation at the binary level and its representation by a symbolic assembler.
- Explain why designers adopt different instruction formats, such as the number of addresses per instruction and variable length versus fixed length formats.
- Create small programs and fragments of assembly language code to demonstrate machine level operations.
- Demonstrate high-level programming constructs at the machine-language level.
- Analyze assembly language programming using computer simulation packages.

Computer arithmetic

- Demonstrate how numerical values are represented in digital computers.
- Identify the limitations of computer arithmetic and discuss the effects of errors on calculations.
- Evaluate the effect of a processor's arithmetic unit on its overall performance.

Memory system organization and architecture

- Identify the main types of memory technology.
- Explain the effect of memory latency and bandwidth on performance.
- Explain the use of memory hierarchy to reduce the effective memory latency.
- Describe the principles of memory management and caching.
- Analyze how errors in memory systems arise and how to resolve them.

Interfacing and communication

- Explain how to use interrupts to implement I/O control and data transfers.
- Create small interrupt service routines and I/O drivers using assembly language.
- Identify various types of buses in a computer system.
- Describe data access from a variety of storage devices.
- Design and construct interfaces.
- Compare and contrast different operating systems.

Circuit Analysis with Lab

This course includes the topics of 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. Experiments introduce basic techniques and problem-solving, while comparisons between theoretical and experimental results are investigated in written laboratory reports.

Prerequisite: Calculus II, Physics I

Syllabus and Student Performance Objectives:

History and overview

- Discuss the achievements of key contributors to the field of circuits and systems.
- Compare and contrast resistance and reactance.
- Apply Ohm's Law to solve problems in circuit analysis.
- Distinguish between inductance and capacitance.
- Describe the characteristics phase.
- Demonstrate aliasing.
- Apply Fourier series to solve problems in circuit analysis.
- Summarize how the concepts of circuits and systems apply to computer engineering.
- Describe how computer engineering uses digital signal processing.

Electrical quantities

- Identify and demonstrate through lab activity basic electrical principles.
- Explain the relationships between basic electrical quantities.

Resistive circuits and networks

- Demonstrate and compose basic resistive circuit equations.
- Analyze and simplify basic resistive circuits.
- Demonstrate network analysis tools for resistive circuits.

Reactive circuits and networks

- Categorize basic energy storage devices.
- Construct various combinations of inductors and capacitors.
- Explain simple transient response of various R, L, and C circuits.
- Analyze and design simple R, L, and C circuits.

Frequency response

- Explain frequency domain characteristics of electrical circuits.
- Analyze and design frequency selective circuits.

Signals

- Describe advantages of group sampling of time signals.
- Contrast the meanings of analog and digital signals.
- Contrast how group size affects signal spectra.
- Identify the advantages of sampling both periodic and non-periodic signals.
- Describe the discrete-time representation of signals.

Sinusoidal analysis

- Investigate response of electrical circuits to sinusoidal signal excitation.
- Analyze circuits using the techniques given.

Appendix B
Discrete Mathematics Course Descriptions

The two discrete mathematics courses discussed in this Appendix are detailed in the *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* report.

CS 105: Discrete Structures I

This course introduces the foundations of discrete mathematics as they apply to computer science, focusing on providing a solid theoretical foundation for further work. Topics include functions, relations, sets, simple proof techniques, Boolean algebra, propositional logic, digital logic, elementary number theory, and the fundamentals of counting.

Prerequisites: Mathematical preparation sufficient to qualify for precalculus at the college level.
Corequisite: CS102

Syllabus:

- Introduction to logic and proofs: Direct proofs; proof by contradiction; mathematical induction
- Fundamental structures: Functions (surjections, injections, inverses, composition); relations (reflexivity, symmetry, transitivity, equivalence relations); sets (Venn diagrams, complements, Cartesian products, power sets); pigeonhole principle; cardinality and countability
- Boolean algebra: Boolean values; standard operations on Boolean values; de Morgan's laws
- Propositional logic: Logical connectives; truth tables; normal forms (conjunctive and disjunctive); validity
- Digital logic: Logic gates, flip-flops, counters; circuit minimization
- Descriptive statistics: methods of collecting data, frequency distribution graphs, measures of central tendency, variation, and position, and use of z-scores.
- Basics of counting: Counting arguments; pigeonhole principle; permutations and combinations; binomial coefficients

Knowledge Area	***Units Covered***	***Suggested Hours***
Discrete Structures	Functions, relations, and sets	9
	Basic logic	5
	Proof techniques	4
	Basics of counting	9
	Interpreting descriptive statistics	9
Architecture	Digital logic and digital systems	3
Social & Professional	Methods and tools of analysis	1
	TOTAL:	40

Table 3: CS 105, Discrete Structures I

Notes:

The Discrete Structures (DS) material is divided into two courses. CS105 covers the first half of the material followed by CS106, which completes the topic coverage. Although the principal focus is discrete mathematics, the course is likely to be more successful if it highlights applications whose solutions require proof, logic, and counting.

CS 106: Discrete Structures II

This course continues the discussion of discrete mathematics introduced in CS 105. Topics in the second course include predicate logic, recurrence relations, graphs, trees, matrices, computational complexity, elementary computability, and discrete probability.

Prerequisite: CS 105
Corequisite: CS 103

Syllabus:

- Review of previous course
- Predicate logic: Universal and existential quantification; modus ponens and modus tollens; limitations of predicate logic
- Recurrence relations: Basic formulae; elementary solution techniques
- Graphs and trees: Fundamental definitions; simple algorithms; traversal strategies; proof techniques; spanning trees; applications
- Matrices: Basic properties; applications
- Computational complexity: Order analysis; standard complexity classes
- Elementary computability: Countability and uncountability; diagonalization proof to show uncountability of the reals; definition of the P and NP classes; simple demonstration of the halting problem
- Discrete probability: Finite probability spaces; conditional probability, independence, Bayes' rule; random events; random integer variables; mathematical expectation

Knowledge Area	***Units Covered***	***Suggested Hours***
Discrete Structures	Basic logic	7
	Proof techniques	8
	Graphs and trees	4
	Discrete probability	6
Algorithms	Basic algorithmic analysis	2
	Basic computability	3
	The complexity classes P and NP	2
Other	Matrices	3
	Topics of local interest	5
	TOTAL:	40

Table 4: CS 106, Discrete Structures II

Notes:
This implementation of the Discrete Structures area (DS) divides the material into two courses: CS105 and CS106. Like CS 105, CS 106 introduces mathematical topics in the context of applications that require those concepts as tools. For this course, likely applications include transportation network problems (such as the traveling salesperson problem) and resource allocation. Matrices have computing applications in many areas, including inventory control, cost analysis, and data analysis. The unit on matrices introduces basic terminology, the operations of addition, scalar and matrix multiplication, the transpose and inverse, and 2x2 and 3x3 determinants.

Appendix C
Computer Science Course Descriptions

The two paradigms discussed in this Appendix are detailed in the *Computing Curricula 2003: Guidelines for Associate-Degree Curricula in Computer Science* report.

Computer Science Imperative-First: Course Descriptions
The Imperative-first approach consists of a three-course sequence that begins with a procedural structured-programming approach to fundamental programming concepts, followed by object-oriented concepts, and culminates with data structures.

CS101I: Programming Fundamentals

This course introduces the fundamental concepts of procedural programming. Topics include data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging. The course also offers an introduction to the historical and social context of computing and an overview of computer science as a discipline.

Syllabus:
- Computing applications: Word processing; spreadsheets; editors; files and directories
- Fundamental programming constructs: Syntax and semantics of a higher-level language; variables, types, expressions, and assignment; simple I/O; conditional and iterative control structures; functions and parameter passing; structured decomposition
- Algorithms and problem-solving: Problem-solving strategies; the role of algorithms in the problem-solving process; implementation strategies for algorithms; debugging strategies; the concept and properties of algorithms
- Fundamental data structures: Primitive types; arrays; records; strings and string processing
- Machine level representation of data: Bits, bytes, and words; numeric data representation and number bases; representation of character data
- Overview of operating systems: The role and purpose of operating systems; simple file management
- Introduction to net-centric computing: Background and history of networking and the Internet; demonstration and use of networking software including e-mail, telnet, and FTP
- Human-computer interaction: Introduction to design issues
- Software development methodology: Fundamental design concepts and principles; structured design; testing and debugging strategies; test-case design; programming environments; testing and debugging tools
- Social context of computing: History of computing and computers; evolution of ideas and machines; social impact of computers and the Internet; professionalism, codes of ethics, and responsible conduct; copyrights, intellectual property, and software piracy.

CS102I: The Object-Oriented Paradigm

This course introduces the concepts of object-oriented programming to students with a background in the procedural paradigm. The course begins with a review of control structures and data types with emphasis on structured data types and array processing. It then moves on to introduce the object-oriented programming paradigm, focusing on the definition and use of classes along with the fundamentals of object-oriented design. Other topics include an overview of programming language principles, simple analysis of algorithms, basic searching and sorting techniques, and an introduction to software engineering issues.

Syllabus:

- Review of control structures, functions, and primitive data types
- Object-oriented programming: Object-oriented design; encapsulation and information-hiding; separation of behavior and implementation; classes, subclasses, and inheritance; polymorphism; class hierarchies
- Fundamental computing algorithms: simple searching and sorting algorithms (linear and binary search, selection and insertion sort)
- Fundamentals of event-driven programming
- Introduction to computer graphics: Using a simple graphics API
- Overview of programming languages: History of programming languages; brief survey of programming paradigms
- Virtual machines: The concept of a virtual machine; hierarchy of virtual machines; intermediate languages
- Introduction to language translation: Comparison of interpreters and compilers; language translation phases; machine-dependent and machine-independent aspects of translation
- Introduction to database systems: History and motivation for database systems; use of a database query language
- Software evolution: Software maintenance; characteristics of maintainable software; reengineering; legacy systems; software reuse

CS103I: Data Structures and Algorithms

This course builds upon the foundation provided by the CS101I-102I sequence to introduce the fundamental concepts of data structures and the algorithms that proceed from them. Topics include recursion, the underlying philosophy of object-oriented programming, fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), the basics of algorithmic analysis, and an introduction to the principles of language translation.

Syllabus:

- Review of elementary programming concepts
- Fundamental data structures: Stacks; queues; linked lists; hash tables; trees; graphs
- Object-oriented programming: Object-oriented design; encapsulation and information hiding; classes; separation of behavior and implementation; class hierarchies; inheritance; polymorphism
- Fundamental computing algorithms: O(N log N) sorting algorithms; hash tables, including collision-avoidance strategies; binary search trees; representations of graphs; depth- and breadth-first traversals
- Recursion: The concept of recursion; recursive mathematical functions; simple recursive procedures; divide-and-conquer strategies; recursive backtracking; implementation of recursion
- Basic algorithmic analysis: Asymptotic analysis of upper and average complexity bounds; identifying differences among best, average, and worst case behaviors; big O, little o, omega, and theta notation; standard complexity classes; empirical measurements of performance; time and space tradeoffs in algorithms; using recurrence relations to analyze recursive algorithms
- Algorithmic strategies: Brute-force algorithms; greedy algorithms; divide-and-conquer; backtracking; branch-and-bound; heuristics; pattern matching and string/text algorithms; numerical approximation algorithms
- Overview of programming languages: Programming paradigms
- Software engineering: Software validation; testing fundamentals, including test plan creation and test case generation; object-oriented testing

Computer Science Objects-First: Course Descriptions

The Objects-first implementation strategy incorporates throughout the curriculum object-oriented software design and programming methodologies. Object-oriented design promotes thinking about software development in a way that more closely models interaction with the real world. Modeling programming problems as abstract objects that communicate with each other helps to manage the complexity of software projects. The object-oriented programming paradigm is based on the relationship of interacting and cooperating data objects to solve computing problems.

CS101O: Introduction to Object-Oriented Programming

This course introduces the fundamental concepts of programming from an object-oriented perspective. Topics include simple data types, control structures, an introduction to array and string data structures and algorithms, as well as debugging techniques and the social implications of computing. The course emphasizes good software engineering principles and developing fundamental programming skills in the context of a language that supports the object-oriented paradigm.

Syllabus:

- Introduction to the history of computer science
- Ethics and responsibility of computer professionals
- Introduction to computer systems and environments
- Introduction to object-oriented paradigm: Abstraction; objects; classes; methods; parameter passing; encapsulation; inheritance; polymorphism
- Fundamental programming constructs: Basic syntax and semantics of a higher-level language; variables, types, expressions, and assignment; simple I/O; conditional and iterative control structures; structured decomposition
- Fundamental data structures: Primitive types; arrays; records; strings and string processing
- Introduction to programming languages
- Algorithms and problem-solving: Problem-solving strategies; the role of algorithms in the problem-solving process; implementation strategies for algorithms; debugging strategies; the concept and properties of algorithms

CS102O: Objects and Data Abstraction

This course continues the introduction from CS101O to the methodology of programming from an object-oriented perspective. Through the study of object design, this course also introduces the basics of human-computer interfaces, graphics, and the social implications of computing, with an emphasis on software engineering.

Syllabus:

- Review of object-oriented programming: Object-oriented methodology, object-oriented design; software tools
- Principles of object-oriented programming: Inheritance; class hierarchies; polymorphism; abstract and interface classes; container/collection classes and iterators
- Object-oriented design: Concept of design patterns and the use of APIs; modeling tools such as class diagrams, CRC cards, and UML use cases
- Virtual machines: The concept of a virtual machine; hierarchy of virtual machines; intermediate languages
- Fundamental computing algorithms: Searching; sorting; introduction to recursive algorithms
- Fundamental data structures: Built-in, programmer-created, and dynamic data structures
- Event-driven programming: Event-handling methods; event propagation; exception handling
- Foundations of human-computer interaction: Human-centered development and evaluation; principles of good design and good designers; engineering tradeoffs; introduction to usability testing
- Fundamental techniques in graphics: Hierarchy of graphics software; using a graphics API; simple color models; homogeneous coordinates; affine transformations; viewing transformation; clipping
- Software engineering issues: Tools; processes; requirements; design and testing; design for reuse; risks and liabilities of computer-based systems

CS103O: Algorithms and Data Structures

This course builds upon the introduction to object-oriented programming begun in CS101O and CS102O with an emphasis on algorithms, data structures, and software engineering.

Syllabus:

- Review of object-oriented design
- Review of basic algorithm design
- Review of professional and ethical issues
- Algorithms and problem solving: Classic techniques for algorithm design; problem solving in the object-oriented paradigm; application of algorithm design techniques to a medium-sized project, with an emphasis on formal methods of testing
- Basic algorithmic analysis: Asymptotic analysis of upper and average complexity bounds; identifying differences among best, average, and worst case behaviors; big O notation; standard complexity classes; empirical measurements of performance; time and space tradeoffs in algorithms
- Recursion: The concept of recursion; recursive mathematical functions; simple recursive procedures; divide-and-conquer strategies; recursive backtracking; implementation of recursion; recursion on trees and graphs
- Fundamental computing algorithms: Hash tables; binary search trees; representations of graphs; depth- and breadth-first traversals; shortest-path algorithms; transitive closure; minimum spanning tree; topological sort
- Fundamental data structures: Pointers and references; linked structures; implementation strategies for stacks, queues, and hash tables; implementation strategies for graphs and trees; strategies for choosing the right data structure
- Software engineering: Software project management; building a medium-sized system, in teams, with algorithmic efficiency in mind

Appendix D
ACM TYC Taxonomy of Learning Processes

Table 5 is an adaptation of Bloom's Taxonomy (1956) by the ACM Two-Year College Education Committee used across all their curricular reports beginning with 1993.

Level of Taxonomy	Definition	Verbs to Help Design Activities
Factual Knowledge	Recall information	Tell - list - define – name – recall - identify - remember – repeat – recognize
Comprehension	Understanding of communicated material or information	Transform - change - restate – describe - explain - interpret – summarize - discuss
Applicative Knowledge	Apply basic rules and conventions	Add – subtract – punctuate – edit – divide – multiply – diagram
Procedural Knowledge	Complete tasks using multi-step processes	Apply – investigate – produce
Analysis	Breaking down information into its parts	Analyze - dissect – distinguish - examine - compare - contrast – survey - categorize
Synthesis	Putting together ideas into a new or unique product	Create – invent – compose – construct - design - produce – modify
Evaluation	Judging the value of materials or ideas based on set standards or criteria	Judge - decide – justify – evaluate - critique - debate – verify – recommend
Higher-Order Thinking	Apply analysis, syntheses and evaluation processes to solve complex problems	Evaluate - create – conduct – analyze
Attitudes and Values	Express feelings, opinions, personal beliefs regarding people, objects and events	Respect – demonstrate – express
Social Behaviors	Learned behavior that conforms to acceptable social standards	Perform – communicate
Motor Skills	Physical coordination, strength, control, skills related to physical tasks	Demonstrate - run – dribble - move - show

Table 5: ACM TYC Taxonomy of Learning Processes