GUIDELINES
for
ASSOCIATE-DEGREE PROGRAMS
in COMPUTER SCIENCE

August 2002

Produced By The
ACM Two-Year College Education Committee

Richard Austing
Robert D. Campbell, Chair
C. Fay Cover
Elizabeth K. Hawthorne
Karl J. Klee

In Collaboration With
Academic and Industry Advisors and Reviewers

Association for Computing Machinery

Institute of Electrical & Electronics Engineers
Approval Pending

Association for Computing Machinery
1515 Broadway, 17th Floor
New York, New York 10036

Copyright © 2002 by the Association for Computing Machinery, Inc. Copying without fee is permitted provided that the copies are not made or distributed for direct commercial advantage, and credits to the source is given. Abstracting with credit is permitted. For permission to republish write to: Director of Publications, Association for Computing Machinery. To copy otherwise, or republish requires a fee and/or specific permission.

ISBN: 1-58113-559-9

PDF and MS Word files are available for download from:
http://www.acmtyc.org/

Additional printed copies may be ordered prepaid from:

ACM Order Department
PO Box 12114
Church Street Station
New York, NY 10257
800-342-9926
+1-212-626-0500

ACM Order Number: 100020

Printed in the USA at Rock Valley College, Rockford, Illinois

TABLE OF CONTENTS

CHAPTER 1. OVERVIEW 1

1.1 GOALS AND PURPOSE 1
1.2 BACKGROUND 1
1.3 STRUCTURE OF THIS REPORT 2
1.4 THE TWO-YEAR COLLEGE ENVIRONMENT 3
1.5 SUSTAINING A PROGRAM 3
1.6 ARTICULATION OF TRANSFER PROGRAMS 4
1.7 CAREER-ORIENTED PROGRAMS 7
1.8 INCORPORATING PROFESSIONAL PRACTICES 8
1.9 ADDITIONAL PROGRAM COMPONENTS 8

CHAPTER 2. BODY OF KNOWLEDGE 10

2.1 INTRODUCTION 10
2.2 ALGORITHMS AND COMPLEXITY (AL) 11
2.3 ARCHITECTURE AND ORGANIZATION (AR) 14
2.4 DISCRETE STRUCTURES (DS) 18
2.5 GRAPHICS AND VISUAL COMPUTING (GV) 22
2.6 HUMAN COMPUTER INTERACTION (HC) 23
2.7 INFORMATION MANAGEMENT (IM) 26
2.8 NET-CENTRIC COMPUTING (NC) 28
2.9 OPERATING SYSTEMS (OS) 33
2.10 PROGRAMMING FUNDAMENTALS (PF) 39
2.11 PROGRAMMING LANGUAGES (PL) 43
2.12 SOFTWARE ENGINEERING (SE) 47
2.13 SOCIAL AND PROFESSIONAL ISSUES (SP) 52

CHAPTER 3. CURRICULA 55

3.1 CURRICULUM OVERVIEW 55
3.2 THE OBJECTS-FIRST IMPLEMENTATION STRATEGY 57
3.3 THE IMPERATIVE-FIRST IMPLEMENTATION STRATEGY 63
3.6 ELECTIVES 79

ACKNOWLEDGEMENTS 88

BIBLIOGRAPHY AND REFERENCES 89

APPENDIX A. TAXONOMY OF LEARNING PROCESSES 90

Chapter 1. Overview

1.1 Goals and Purpose

This report provides guidelines for computer science programs in associate-degree granting institutions. It places a principle focus on programs designed for students intending to transfer into baccalaureate programs, accompanied by deliberate guidelines designed to facilitate matters of articulation. It also presents a body of knowledge for computer science in the two-year college setting, associated learning objectives, and detailed descriptions of computer science and accompanying mathematics courses appropriate for implementation in associate-degree programs.

Three specific implementation approaches for introductory courses provide the backbone for this report: imperative-first, objects-first, and breadth-first. The report specifies computer science elective courses that have a computer science introductory course sequence as a prerequisite; these electives provide curricula to support a career-oriented graduate as well as content appropriate for transfer consideration. In addition, it outlines required mathematics courses. It addresses other aspects requisite to a successful computer science program, including a brief indication of the two-year college environment; administration, faculty, and computing resource issues; and requirements from other disciplines.

This report updates the Computing Science volume of the Computing Curricula Guidelines for Associate-Degree Programs issued in 1993 by the ACM Two-Year College Education Committee. The report also shares common goals and outcomes with the recent computer science curricula recommendations for baccalaureate programs developed by the Joint Task Force on Computing Curricula 2001 established by the Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS) and the Association for Computing Machinery (ACM). The IEEE-CS and ACM recognized the need to revise and update their previous curricular reports; the new *CC2001 CS report* can be found at http://www.computer.org.

1.2 Background

In the spring of 2001, the ACM Two-Year College Education Committee assembled a joint IEEE-CS/ACM Task Force made up of computing faculty, administrators and industry representatives to prepare curricular guidelines for associate-degree programs in computer science. This Task Force split into subgroups based on implementation strategies, and used the Iron-Man version of the draft IEEE-CS/ACM CC2001 Report, as well as the two-year college 1993 Report, as a starting point from which to identify the topics and objectives that should be included in an associate-degree computer science program. The subgroups subsequently assembled this work into a draft report and shared it with the entire Task Force and with the CC2001 Steering Committee for feedback. In the fall of 2001, a sub-committee of the Task Force further refined the objectives, revised the courses, composed the narrative, and distributed internationally the entire document to a set of reviewers. Following that review, the ACM Two-Year College Education

Committee produced the final report, which the IEEE Computer Society Education Activities Board and the ACM Education Board subsequently accepted and published.

1.3 Structure of this Report

This report contains three chapters: Overview, Body of Knowledge, and Curricula. It discusses the general issues related to offering a computer science program in a two-year college environment in this Overview. The Body of Knowledge chapter specifies the following areas, or sub-fields, of computer science:

- Algorithms and Complexity (AL)
- Architecture and Organization (AR)
- Discrete Structures (DS)
- Graphics and Visual Computing (GV)
- Human-Computer Interaction (HC)
- Information Management (IM)
- Net-Centric Computing (NC)
- Operating Systems (OS)
- Programming Fundamentals (PF)
- Programming Languages (PL)
- Software Engineering (SE)
- Social and Professional Issues (SP)

Each of these areas consists of units. In turn, each unit consists of topics with associated learning objectives.

The Curricula chapter defines three approaches for implementing a computer science program; for each, sample courses, both required and elective, are detailed. Each course description includes a list of topics with the appropriate associated learning objectives. This chapter also includes a chart and table illustrating the course sequencing.

The three strategies for implementation are: imperative-first, objects-first, and breadth-first.

- 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.

- The Objects-first approach incorporates object-oriented software design and implementation methodologies throughout the curriculum. Object-oriented programming uses the relationship of interacting and cooperating data objects to solve computing problems.

- The Breadth-first approach covers a wide expanse of computer science topics in a three-course sequence, and emphasizes the mastery, selection and use of appropriate programming paradigms in a problem-solving context.

1.4 The Two-Year College Environment

The two-year college environment is a unique setting due to a variety of factors resulting from the threefold mission of two-year colleges 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 and the sources of a wide range of community events, activities, and services.

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.

1.5 Sustaining a Program

Administration and leadership at two-year colleges must be prepared to provide adequate support for computer science programs, often within an environment very limited resources. Two-year colleges must fund the resources and professional activities necessary to sustain these programs, including:

- a sufficient number of qualified faculty;
- a plan for development, implementation, evaluation and revision of curriculum by faculty;
- the opportunity for ongoing professional growth and development of faculty and support staff;
- an industry advisory committee empowered to influence decisions and impact the program;
- state-of-the-art computer classrooms, labs and facilities, dedicated to specialized instruction in computer science courses;
- staffing support for infrastructure needs and end-user support for students and faculty; and
- periodic updating, enhancing and replacing computing hardware and software;

Recruiting, hiring, and retaining qualified fulltime faculty continues to be a challenge for two-year colleges. It is necessary to make extensive use of adjunct and part-time faculty, who fortunately, often provide specific technical skills and teaching techniques for specialized courses. However, the recruitment, development, evaluation, and retention

of talented adjunct faculty require a commitment of institutional financial resources, as well as the time of faculty and staff.

Two-year colleges must accurately and effectively measure student learning with associated outcomes. Accreditation requirements, performance-based funding and public demands for accountability have made effective educational assessment a necessity. Hence, computer science courses and programs must have clear, precise and measurable learning objectives with associated student outcomes.

Two-year colleges must develop transition and articulation strategies applicable to the colleges and universities to which their students most often transfer. It may be necessary to modify course content to facilitate transfer credit and articulation agreements. In some cases, it may be necessary for two-year colleges to modify their list of recommended courses to match the requirements of their primary transfer institutions. The Task Force designed the courses recommended in this report, especially those specified for a beginning sequence, to facilitate articulation between a two-year college and a baccalaureate institution. Programs must take into consideration the general education requirements at both the two-year college and the transfer institution in the formulation of a student's program of study.

Many students enter two-year colleges with insufficient mathematics preparation for a computer science program. Such students must devote additional semesters to achieve the mathematical maturity and problem-solving skills required to be successful in computer science coursework. The two-year college programs of study outlined in this report expect that students have sufficient mathematics background and computer fluency skills prior to entering the first computing course. These programs also require two semesters of discrete mathematics as integrated components in the overall degree requirements. Furthermore, the studies of computing and mathematics require coordination and synchronization to ensure that students engage theory and application in a meaningful way.

1.6 Articulation of Transfer Programs

Articulation of courses and programs between two academic institutions facilitates the transfer of students from one institution to the other. 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.

Students must realize that courses and program requirements change over time. If a student takes too long to complete a program, a different articulation agreement might be in effect at the time a student requests to transfer. Hence, students must expect to complete programs 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.

Faculty must ensure that they clearly define program requirements, address program objectives in a responsible manner, and evaluate 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. This report is specifically designed to promote articulation by enabling computer science faculty in two-year colleges and universities to compare curricula on a topical basis using this report together with the new IEEE-CS/ACM CC2001 Computer Science Report. The table below lists the computer science body of knowledge (areas and units) from the *CC2001 CS Report*; it can be seen that the two reports share area and unit headings where appropriate. The italicized unit headings represent upper-division areas and units that are not included in the two-year college recommendations.

Table 1.1 Computer Science Body of Knowledge (CC2001 CS Report)

DS. Discrete Structures DS1. Functions, relations, and sets DS2. Basic logic DS3. Proof techniques DS4. Basics of counting DS5. Graphs and trees DS6. Discrete probability **PF. Programming Fundamentals** PF1. Fundamental programming constructs PF2. Algorithms and problem-solving PF3. Fundamental data structures PF4. Recursion PF5. Event-driven programming **AL. Algorithms and Complexity** AL1. Basic algorithmic analysis AL2. Algorithmic strategies AL3. Fundamental computing algorithms *AL4. Distributed algorithms* AL5. Basic computability theory AL6. The complexity classes P and NP *AL7. Automata theory* *AL8. Advanced algorithmic analysis* *AL9. Cryptographic algorithms* *AL10. Geometric algorithms* *AL11. Parallel algorithms* **PL. Programming Languages** PL1. Overview of programming languages PL2. Virtual machines PL3. Introduction to language translation PL4. Declaration and types PL5. Abstraction mechanisms PL6. Object-oriented programming PL7. Functional programming *PL8. Language translation systems* *PL9. Type systems* *PL10. Programming language semantics* *PL11. Programming language design*	**GV. Graphics and Visual Computing** GV1. Fundamental techniques in graphics GV2. Graphic systems *GV3. Graphic communication* *GV4. Geometric modeling* *GV5. Basic rendering* *GV6. Advanced rendering* *GV7. Advanced techniques* *GV8. Computer animation* *GV9. Visualization* *GV10. Virtual reality* *GV11. Computer vision* **IS. Intelligent Systems** *IS1. Fundamental issues in intelligent systems* *IS2. Search and constraint satisfaction* *IS3. Knowledge representation and reasoning* *IS4. Advanced search* *IS5. Advanced knowledge representation and reasoning* *IS6. Agents* *IS7. Natural language processing* *IS8. Machine learning and neural networks* *IS9. AI planning systems* *IS10. Robotics* **IM. Information Management** IM1. Information models and systems IM2. Database systems *IM3. Data modeling* *IM4. Relational databases* IM5. Database query languages *IM6. Relational database design* *IM7. Transaction processing* *IM8. Distributed databases* *IM9. Physical database design* *IM10. Data mining* *IM11. Information storage and retrieval* IM12. Hypertext and hypermedia *IM13. Multimedia information and systems* *IM14. Digital libraries*

continued

Table 1.1 Computer Science Body of Knowledge continued

AR. Architecture and Organization AR1. Digital logic and digital systems AR2. Machine level representation of data AR3. Assembly level machine organization AR4. Memory system organization and architecture AR5. Interfacing and communication AR6. Functional organization AR7. Multiprocessing and alternative architectures *AR8. Performance enhancements* AR9. Architecture for networks and distributed systems **OS. Operating Systems** OS1. Overview of operating systems OS2. Operating system principles OS3. Concurrency OS4. Scheduling and dispatch OS5. Memory management OS6. Device management OS7. Security and protection OS8. File systems *OS9. Real-time and embedded systems* *OS10. Fault tolerance* *OS11. System performance evaluation* OS12. Scripting **NC. Net-Centric Computing** NC1. Introduction to net-centric computing NC2. Communication and networking NC3. Network security NC4. The Web as an example of client-server computing NC5. Building web applications NC6. Network management NC7. Compression and decompression NC8. Multimedia data technologies *NC9. Wireless and mobile computing*	**SE. Software Engineering** SE1. Software design SE2. Using APIs SE3. Software tools and environments SE4. Software processes SE5. Software requirements and specifications SE6. Software validation SE7. Software evolution SE8. Software project management *SE9. Component-based computing* *SE10. Formal methods* *SE11. Software reliability* *SE12. Specialized systems development* **SP. Social and Professional Issues** SP1. History of computing SP2. Social context of computing SP3. Methods and tools of analysis SP4. Professional and ethical responsibilities SP5. Risks and liabilities of computer-based systems SP6. Intellectual property *SP7. Privacy and civil liberties* *SP8. Computer crime* *SP9. Economic issues in computing* *SP10. Philosophical frameworks* **CN. Computational Science and Numerical Methods** *CN1. Numerical analysis* *CN2. Operations research* *CN3. Modeling and simulation* *CN4. High-performance computing* **HC. Human-Computer Interaction** HC1. Foundations of human-comupter interaction HC2. Building a simple graphical user interface *HC3. Human-centered software evaluation.* *HC4. Human-centered software development* HC5. Graphical user-interface design *HC6. Graphical user-interface programming* HC7. HCI aspects of multimedia systems. *HC8. HCI aspects of collaboration and communication*

Note: The *italicized* items are not included within the two-year college programs outlined in this report.

In many cases, one particular course in one institution might not be equivalent to a single course at the second institution. However, a group or sequence of courses could be determined equivalent to another course grouping or sequence even though the number of courses differ. In particular, each of the three-course sequences in each of the three approaches in this report is fundamentally equivalent to the corresponding sequences in the IEEE-CS/ACM *CC2001 CS Report*, and each is designed to be at least equivalent to the corresponding two-course sequence in the *CC2001 CS Report*. Students completing any of the three-course sequences described herein should be well prepared for further work in computer science.

1.7 Career-Oriented Programs

Programs at two-year colleges fall into two general orientations, career or transfer. A career program typically provides a broad educational foundation as well as specific knowledge, skills and abilities needed to proceed directly into the work environment. Students graduating from a two-year career-oriented program often enter the work force immediately, and once they have gained work experience, some may then pursue a baccalaureate program in computer science. Unless a college specifically designed a student's associate-degree program for transfer to a baccalaureate program, the student will normally be required to take additional courses before entry into the upper division. At the beginning of their associate-degree studies, the college should make students aware of the distinctions between career and transfer programs, the academic requirements of each, and the associated employment options.

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

- An active computer science industry advisory committee, consisting of employers interested in hiring computer science associate degree graduates. This advisory board should provide guidance concerning the knowledge, skills, and abilities the students must possess to directly enter a career in computer science, within their community.
- Real world work experience including co-op programs, internships and/or other practicum activities, with an emphasis on professional practices.
- A rigorous computer science introductory sequence providing a strong foundation.
- Additional state of the art elective coursework as recommended by the advisory committee.
- Potential articulation paths that enable the career-oriented student to pursue a baccalaureate degree in the future after working for some period of time.
- An assessment process whereby students can earn credit for relevant experience.

This report includes both required and elective courses intended to prepare students either to enter the workforce directly or to transfer into a baccalaureate program. Detailed course descriptions for the required courses (CS101, CS102, CS103) for each implementation as well as the following computer science elective courses are included in the Curricula chapter of this report.

- Introduction to Web-Centric Computing
- Net-Centric Operating Systems
- Computer Organization and Architecture
- Hardware Fundamentals

1.8 Incorporating Professional Practices

A transfer program or a career-oriented program in computer science must provide students a variety of opportunities for professional practices. Faculty at two-year colleges are well aware of the importance of including practical and applied work as an integral part of all computing programs, including computer science. Their students are encouraged to work in teams, solve practical problems in course projects, make presentations, confront issues of privacy and ethics, use current technology in laboratories, and attain real-world experience through cooperative education, internships and other practicum activities. An active industry advisory committee can be an important asset in helping faculty incorporate current professional practices into the curriculum, as well as providing valuable contacts for students seeking employment placement. Faculty know that a conscious and proactive incorporation of professional practices into a curriculum in computer science benefits students, either as a valuable component in a transfer-oriented computer science degree program, or in addressing industry needs for qualified personnel as they exit a career-oriented computer science degree program.

1.9 Additional Program Components

In addition to the required and elective computer science courses, a college must design a degree program to fulfill other objectives as well. These include providing students with a level of mathematical knowledge and ability, familiarity with the scientific method of discovery and reasoning, effective written and oral communication skills, and the ability to work cooperatively and effectively as a member of a team. Prior to entry into the computer science associate-degree program, students must demonstrate mastery of mathematics equivalent to that required to qualify for the standard precalculus course, fundamental computer fluency, and readiness to pursue the college-level coursework called for herein.

The crucial role of mathematics in the foundation of computer science requires that students initiate their mathematics studies early in the pursuit of this degree program. In parallel with their three-course introductory computer science sequence, students must complete a two-semester discrete math sequence (defined in Chapter Three of this report). The learning objectives associated with discrete mathematics specifically support this degree program, and a mathematics department or a computing department (or jointly) should deliver such content with that intent. The discrete mathematics should be an infusion with computer science applications and instructors should continuously reinforce such topics in subsequent computer science courses. Note that no calculus course appears in these degree specifications; however, some baccalaureate programs may continue to designate a calculus requirement, and therefore some transfer students

will need to complete such courses. Many students will also benefit from a course in statistics (see definition in Chapter Three) and this should be available as a program elective.

A familiarity with 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) is extremely valuable in the setting of computer science studies. In fact, the scientific method reasonably presents a basic methodology for much of the discipline of computing. Students pursuing a computer science curriculum should engage rigorous science coursework that includes strong training in the tenets of the scientific method and includes direct hands-on laboratory experiences. Advisors should guide students intending to transfer into a baccalaureate program (immediately or as a long-term goal) to select science coursework with that objective in mind.

Effective communication abilities are of critical importance to nearly every computer science career. Therefore, communication skills must be identified, developed, nurtured and called upon throughout a computer science degree program. A student must master effective writing, speaking, and listening abilities, and then consistently demonstrate those abilities in a variety of settings, including formal and informal, large group and one-on-one, technical and non-technical, point and counter-point. Such activities should occur across the breadth of the curriculum and in particular, should substantially appear in computer science courses.

These guidelines call for the incorporating into the computing curriculum both a breadth of professional practices as well as various forms of effective communication by the students. Programs best serve the students by requiring them to take 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 greater emphasis on interpersonal skills. Students should engage courses as part of their total program that assist them in preparing for this world; in like fashion, such topics should be widely incorporated into numerous courses across many disciplines.

Colleges will 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 compatible with those requirements. Articulation agreements often guide curriculum content as well, and are important considerations in the formulation of programs of study, especially for transfer-oriented programs. For that reason, the Task Force has made every effort to establish compatibility between these guidelines and newly established guidelines for baccalaureate programs in computer science. Faculty interests and areas of expertise also influence programs. Thus, it has made provisions to accommodate specialized emphases within these curriculum guidelines. Industry advisory committees also influence and guide curricula, especially career-oriented programs. Those considerations can generally be fit into these guidelines as well, particularly as a collection of elective courses.

Chapter 2. Body of Knowledge

2.1 Introduction

This chapter details the specific content areas, units, topics and learning objectives for the required computing courses and the mathematics courses in associate-degree programs in computer science. The *Range of coverage time* for each unit is determined by the instructional paradigm (Objects, Imperative, or Breadth) and the chosen electives.

Table 2.1 – Summary of Content Areas and Units

AL. Algorithms and Complexity
- Basic algorithmic analysis
- Algorithmic strategies
- Fundamental computing algorithms
- Basic computability
- The complexity classes P and NP

AR. Architecture and Organization
- Digital logic and digital systems
- Machine level representation of data
- Assembly level machine organization
- Memory system organization and architecture
- Interfacing and communication
- Functional organization
- Multiprocessing and alternative architectures
- Architecture for networks and distributed systems

DS. Discrete Structures
- Functions, relations, and sets
- Basic logic
- Proof techniques
- Basics of counting
- Graphs and trees
- Discrete probability
- Interpreting descriptive statistics

GV. Graphics and Visual Computing
- Fundamental techniques in graphics
- Graphic systems

HC. Human Computer Interaction
- Foundations of human computer interaction
- Building a simple graphical user interface
- Graphical user interface design
- HCI aspects of multimedia systems

IM. Information Management
- Information models and systems
- Database systems
- Database query languages
- Hypertext and hypermedia

NC. Net-Centric Computing
- Introduction to net-centric computing
- Communication and networking
- Network security
- The Web as an example of client-server computing
- Building web applications
- Network management
- Compression and decompression
- Multimedia data technologies

OS. Operating Systems
- Overview of operating systems
- Operating system principles
- Concurrency
- Scheduling and dispatch
- Memory management
- Device management
- Security and protection
- File systems
- Scripting

PF. Programming Fundamentals
- Fundamental programming constructs
- Algorithms and problem-solving
- Fundamental data structures
- Recursion
- Event-driven programming

PL. Programming Languages
- Overview of programming languages
- Virtual machines
- Introduction to language translation
- Declaration and types
- Abstraction mechanisms
- Object-oriented programming
- Functional programming

SE. Software Engineering
- Software design
- Using APIs
- Software tools and environments
- Software processes
- Software requirements and specifications
- Software validation
- Software evolution
- Software project management

SP. Social and Professional Issues
- History of computing
- Social context of computing
- Methods and tools of analysis
- Professional and ethical responsibilities
- Risks and liabilities of computer-based systems
- Intellectual property

2.2 Algorithms and Complexity (AL)

- **Basic algorithmic analysis**
- **Algorithmic strategies**
- **Fundamental computing algorithms**
- **Basic computability**
- **The complexity classes P and NP**

Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends on only two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect.

An important part of computing is the ability to select algorithms appropriate to particular purposes and to apply them, recognizing the possibility that no suitable algorithm may exist. This facility relies on understanding the range of algorithms that address an important set of well-defined problems, recognizing their strengths and weaknesses, and their suitability in particular contexts. Efficiency is a pervasive theme throughout this area.

AL – Basic algorithmic analysis

Range of coverage time: 4-8 hours
Topics:

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

Learning objectives:

1. Explain the use of big O, omega, and theta notation to describe the amount of work done by an algorithm.
2. Use big O, omega, and theta notation to give asymptotic upper, lower, and tight bounds on time and space complexity of algorithms.
3. Determine the time and space complexity of simple algorithms.
4. Deduce recurrence relations that describe the time complexity of recursively defined algorithms.
5. Solve elementary recurrence relations.

AL - Algorithmic strategies

Range of coverage time: 3-6 hours
Topics:

Brute-force algorithms
Greedy algorithms
Divide-and-conquer
Backtracking
Branch-and-bound
Heuristics
Pattern matching and string/text algorithms
Numerical approximation algorithms

Learning objectives:

1. Describe the shortcoming of brute-force algorithms.
2. Implement a greedy algorithm, such as Prim's algorithm.
3. Implement a divide and conquer algorithm like quicksort.
4. Use backtracking to solve problem like mazes.
5. Discuss assorted heuristic problem solving methods.
6. Use pattern matching to analyze substrings.
7. Use numerical approximation to solve mathematical problems, such as finding the roots of a polynomial.

AL - Fundamental computing algorithms

Range of coverage time: 6-8 hours
Topics:

Simple numerical algorithms
Sequential and binary search algorithms
Quadratic sorting algorithms (selection, insertion)
O(N log N) sorting algorithms (Quicksort, heapsort, mergesort)
Hash tables, including collision-avoidance strategies
Binary search trees
Representations of graphs (adjacency list, adjacency matrix)
Depth- and breadth-first traversals
Shortest-path algorithms (Dijkstra's and Floyd's algorithms)
Transitive closure (Floyd's algorithm)
Minimum spanning tree (Prim's and Kruskal's algorithms)
Topological sort

Learning objectives:

1. Design and implement various quadratic and O(NlogN) sorting algorithms.
2. Design and implement an appropriate hashing function for an application.
3. Discuss the efficiency considerations for sorting searching and hashing.

4. Design and implement a collision-resolution algorithm for a hash table.
5. Discuss other performance considerations such as small versus large files, programming time, etc.
6. Solve problems using the fundamental graph algorithms, including depth-first and breadth-first search, single-source and all-pairs shortest paths, transitive closure, topological sort, and at least one minimum spanning tree algorithm.
7. Demonstrate the following abilities: to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in programming context.
8. Implement a minimum spanning tree algorithm.

AL - Basic computability

Range of coverage time: 3-4 hours
Topics:

Finite-state machines
Context-free grammars
Tractable and intractable problems
Uncomputable functions
The halting problem
Implications of uncomputability

Learning objectives:

1. Discuss the concept of finite state machines.
2. Explain context-free grammars.
3. Design a deterministic finite-state machine to accept a specified language.
4. Provide a sample problem that has no algorithmic solution.
5. Provide examples that illustrate the implications of uncomputability.

AL - The complexity classes P and NP

Range of coverage time: 2 hours

Topics:

Definition of the classes P and NP
NP-completeness (Cook's theorem)
Standard NP-complete problems
Reduction techniques

Learning objectives:

1. Define the classes P and NP.
2. Explain the significance of NP-completeness.
3. Prove that a problem is NP-complete by reducing a classic known NP-complete problem to it.

2.3 Architecture and Organization (AR)

- **Digital logic and digital systems**
- **Machine level representation of data**
- **Assembly level machine organization**
- **Memory system organization and architecture**
- **Interfacing and communication**
- **Functional organization**
- **Multiprocessing and alternative architectures**
- **Architecture for networks and distributed systems**

The computer lies at the heart of computing. Without it most of the computing disciplines today would be a branch of theoretical mathematics. To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic. All students of computing should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions. There are practical implications as well. Students need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine. In selecting a system to use, they should to able to understand the tradeoff among various components, such as CPU clock speed versus memory size.

AR - Digital logic and digital systems

Range of coverage time: 1-6 hours
Topics:

Overview and history of computer architecture
Fundamental building blocks (logic gates, flip-flops, counters, registers, PLA)
Logic expressions, minimization, sum of product forms
Register transfer notation
Physical considerations (gate delays, fan-in, fan-out)

Learning objectives:

1. Describe the progression of computer architecture from vacuum tubes to VLSI.
2. Demonstrate an understanding of the basic building blocks with respect to the historical development of computer architecture.
3. Use mathematical expressions to describe the functions of simple combinational and sequential circuits.
4. Design a simple circuit using the fundamental building blocks.
5. Discuss the physical limitations of electronic circuits.

AR - Machine level representation of data

Range of coverage time: 3-5 hours
Topics:

Bits, bytes, and words
Numeric data representation and number bases
Fixed- and floating-point systems
Signed and twos-complement representations
Representation of nonnumeric data (character codes, graphical data)
Representation of records and arrays

Learning objectives:

1. Explain the purpose of different formats to represent numerical data.
2. Explain how negative integers are stored in sign-magnitude and two's-complement representation.
3. Convert numerical data from one base format to another.
4. Discuss how fixed-length number representations affect accuracy and precision.
5. Describe the internal representation of non-numeric data.
6. Describe the internal representation of strings, arrays, and records.

AR - Assembly level machine organization

Range of coverage time: 0-9 hours
Topics:

Basic organization of the von Neumann machine
Control unit; instruction fetch, decode, and execution
Instruction sets and types (data manipulation, control, I/O)
Assembly/machine language programming
Instruction formats
Addressing modes
Subroutine call-and-return mechanisms
I/O and interrupts

Learning objectives:

1. Explain the organization of the classical von Neumann machine and its major functional units.
2. Explain how to execute an instruction in a classical von Neumann machine.
3. Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler.
4. Explain different instruction formats, such as addresses per instruction and variable length vs. fixed length formats.
5. Write, assemble, and execute assembly language programs.
6. Demonstrate how to implement fundamental high-level programming constructs at the assembly and machine language level.

7. Explain how to handle subroutine calls at the assembly level.
8. Explain the basic concepts of interrupts and I/O operations.

AR - Memory system organization and architecture

Range of coverage time: 0-5 hours
Topics:

Storage systems and their technology
Coding, data compression, and data integrity
Memory hierarchy
Main memory organization and operations
Latency, cycle time, bandwidth, and interleaving
Cache memories (address mapping, block size, replacement and store policy)
Virtual memory (page table, TLB)
Fault handling and reliability

Learning objectives:

1. Identify the main types of memory technology.
2. Explain the effect of memory latency on running time.
3. Explain the use of memory hierarchy to reduce the effective memory latency.
4. Describe the principles of memory management.
5. Describe the role of cache and virtual memory.
6. Explain the workings of a system with virtual memory management.

AR - Interfacing and communication

Range of coverage time: 0-3 hours
Topics:

I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O
Interrupt structures: vectored and prioritized, interrupt acknowledgment
External storage, physical organization, and drives
Buses: bus protocols, arbitration, direct-memory access (DMA)
Introduction to networks
Multimedia support
RAID architectures

Learning objectives:

1. Explain how to use interrupts to implement I/O control and data transfers.
2. Identify various types of buses in a computer system.
3. Describe data access from a magnetic disk drive and the advantages of RAID architecture.
4. Compare the common network configurations.
5. Identify interfaces needed for multimedia support.

AR - Functional organization

Range of coverage time: 0-3 hours
Topics:

Implementation of simple datapaths
Control unit: hardwired realization vs. microprogrammed realization
Instruction pipelining
Introduction to instruction-level parallelism (ILP)

Learning objectives:

1. Compare alternative implementation of datapaths.
2. Discuss the concept of control points and the generation of control signals using hardwired or microprogrammed implementations.
3. Explain basic instruction level parallelism using pipelining and the major hazards that may occur.

AR - Multiprocessing and alternative architectures

Range of coverage time: 0-3 hours
Topics:

Introduction to SIMD, MIMD, VLIW, EPIC
Systolic architecture
Interconnection networks (hypercube, shuffle-exchange, mesh, crossbar)
Shared memory systems
Cache coherence
Memory models and memory consistency

Learning objectives:

1. Discuss the concept of parallel processing beyond the classical von Neumann model.
2. Describe alternative architectures such as SIMD, MIMD, and VLIW.

AR - Architecture for networks and distributed systems

Range of coverage time: 0-7 hours

Topics:

Introduction to LANs and WANs
Layered protocol design, ISO/OSI, IEEE 802
Impact of architectural issues on distributed algorithms
Network computing
Distributed multimedia

Learning Objectives:

1. Explain the basic components of network systems and distinguish between LANs and WANs.

2. Demonstrate an understanding of protocol design using the OSI and DoD models.
3. Explain how architectures differ in network and distributed systems.
4. Discuss issues related to network computing and distributed multimedia.

2.4 Discrete Structures (DS)

- Functions, relations, and sets
- Basic logic
- Proof techniques
- Basics of counting
- Graphs and trees
- Discrete probability
- Interpreting Descriptive Statistics

The area of discrete structures is foundational material for computer science. By *foundational* we mean that relatively few computer scientists will be working primarily on discrete structures, but that many other areas of computer science require the ability to work with concepts from discrete structures. The area of discrete structures includes important material from such areas as set theory, logic, graph theory, and combinatorics. The notion of formal, mathematical proof is a unifying theme throughout the area.

The material in discrete structures is pervasive in the areas of data structures and algorithms but appears elsewhere in computer science as well. 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.

As the field of computer science matures, increasingly sophisticated analysis techniques are being brought to bear on practical problems. To understand the computational techniques of the future, today's students will need a strong background in discrete structures.

Finally, we note that while areas often have somewhat fuzzy boundaries, this is especially true for discrete structures. The Task Force has gathered together here a body of material of a mathematical nature that computer science education must include, and that computer science educators know well enough to specify in detail. However, the decision about where to draw the line between this area and the Algorithms and Complexity area (AL) on the one hand, and topics left only as supporting mathematics on the other hand, was inevitably somewhat arbitrary. We remind readers that there are vital topics from those two areas that some schools will include in courses with titles like discrete structures.

DS - Functions, relations, and sets

Range of coverage time: 9 hours
Topics:

Functions (surjections, injections, inverses, composition)
Relations (reflexivity, symmetry, transitivity, equivalence relations)
Sets (Venn diagrams, complements, Cartesian products, power sets)
Pigeonhole principle
Cardinality and countability

Learning objectives:

1. Explain with examples the basic terminology of functions, relations, and sets.
2. Perform the operations associated with sets, functions, and relations.
3. Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
4. Demonstrate basic counting principles, including uses of diagonalization and the pigeonhole principle.

DS - Basic logic

Range of coverage time: 12 hours
Topics:

Propositional logic
Logical connectives
Truth tables
Normal forms (conjunctive and disjunctive)
Validity
Predicate logic
Universal and existential quantification
Modus ponens and modus tollens
Limitations of predicate logic

Learning objectives:

1. Apply formal methods of symbolic propositional and predicate logic.
2. Recognize how formal tools of symbolic logic are used to model algorithms and real-life situations.
3. Use formal logic proofs and logical reasoning to solve problems such as puzzles.
4. Recognize the importance and limitations of predicate logic.

DS - Proof techniques

Range of coverage time: 12 hours
Topics:

Notions of implication, converse, inverse, contrapositive, negation, and contradiction
The structure of formal proofs
Direct proofs
Proof by counterexample
Proof by contraposition
Proof by contradiction
Mathematical induction
Strong induction
Recursive mathematical definitions
Well orderings

Learning objectives:

1. Outline the basic structure of and give examples of each proof technique.
2. Discuss which type of proof is best for a given problem.
3. Relate the ideas of mathematical induction to recursion and recursively defined structures.
4. Identify the difference between mathematical and strong induction and give examples of the appropriate use of each.
5. Apply proof techniques to solve problems in computer science, including software engineering, program semantics, and algorithm analysis.

DS - Basics of counting

Range of coverage time: 9 hours
Topics:

Counting arguments
The pigeonhole principle
Permutations and combinations
Solving recurrence relations (common examples, the Master Theorem)

Learning objectives:

1. Apply the sum and product rule for counting.
2. Explain the difference between combinations, C(n, r) and permutations, P(n, r).
3. Discuss Pascal's Identity for combinations and the Binomial Theorem.
4. Describe the concepts of initial condition, recurrence relation and the solution of recurrence relation.
5. Solve problems using arithmetic progressions, geometric progressions, and Fibonacci numbers.
6. Use the principle of inclusion-exclusion for counting.
7. Apply the pigeonhole principle.

8. Discuss examples of recurrence, including matrix multiplication, triangulation, and sorting.
9. Demonstrate the use of the Master Theorem to provide an instant asymptotic solution.
10. Distinguish among various cases of the Master Theorem.

DS - Graphs and trees

Range of coverage time: 4 hours
Topics:

Trees
Undirected graphs
Directed graphs
Spanning trees
Traversal strategies

Learning objectives:

1. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each.
2. Examine different traversal methods for trees and graphs.
3. Model problems in computer science using graphs and trees.
4. Relate graphs and trees to data structures, algorithms, and counting.

DS - Discrete probability

Range of coverage time: 6 hours
Topics:

Finite probability space, probability measure, events
Conditional probability, independence, Bayes' rule
Integer random variables, expectation

Learning objectives:

1. Calculate probabilities of events and expectations of random variables for elementary problems such as games of chance.
2. Differentiate between dependent and independent events.
3. Apply the Binomial Theorem to independent events and Bayes Theorem to dependent events.
4. Apply the tools of probability to solve problems such as the Monte Carlo method, the average case analysis of algorithms, and hashing.

DS – Interpreting descriptive statistics

Range of coverage time: 9 hours

Topics:

Inferential and descriptive statistics;
Data types – numerical and qualitative data;
Methods of collecting data, polls;
Frequency distributions graphs – histogram, pie graphs;
Shapes of distributions – symmetric, skewed right and left;
Computation and use of measures of central tendency (mean, median, mode), variation (standard deviation, range), and position (percentiles, standard score);
Use of the z-score with the normal distribution – comparing data from different distributions, computing percentiles;
Using technology to graph distributions and compute descriptive statistics

Learning objectives:

1. Distinguish between descriptive and inferential statistics and discuss the problems with using entire populations to obtain data.
2. Differentiate between numeric and qualitative data and review the graphing methods and statistics appropriate to each.
3. Use technology (e.g., Minitab, the TI-83 or Excel) to compute statistics and graph distributions.
4. Calculate and discuss the appropriate use of different measures of central tendency, variation, and position.
5. Discuss examples of the misuse of statistics.
6. Solve normal distribution problems, using z-scores and a table or technology.

2.5 Graphics and Visual Computing (GV)

- **Fundamental techniques in graphics**
- **Graphic systems**

Computer graphics is the art and science of communicating information. It uses images generated and presented through computation. This requires the design and construction of models that represent information in ways that support the creation and viewing of images, the design of devices and techniques through which the person may interact with the model or the view, the creation of techniques for rendering the model, and the design of ways the images may be preserved. The goal of computer graphics is to engage the person's visual centers alongside other cognitive centers in understanding.

GV - Fundamental techniques in graphics

Range of coverage time: 1-2 hours
Topics:

Hierarchy of graphics software
Using a graphics API
Simple color models (RGB, HSB, CMYK)
Homogeneous coordinates
Affine transformations (scaling, rotation, translation)
Viewing transformation
Clipping

Learning objectives:

1. Distinguish the capabilities of different levels of graphics software and describe the appropriateness of each.
2. Create images using a standard graphics API.
3. Use the facilities provided by a standard API to express basic transformations such as scaling, rotation, and translation.
4. Implement simple procedures that perform transformation and clipping operations on a simple 2-dimensional image.
5. Discuss the 3-dimensional world coordinate system.

GV - Graphic systems

Range of coverage time: 0-1 hours
Topics:

Raster and vector graphics systems
Video display devices
Physical and logical input devices
Issues facing the developer of graphical systems

Learning objectives:

1. Describe the appropriateness of graphics architecture for a given application.
2. Explain the functions of different input devices.

2.6 Human Computer Interaction (HC)

- **Foundations of human-computer interaction**
- **Building a simple graphical user interface**
- **Graphical user-interface design**
- **HCI aspects of multimedia systems**

This list of topics presents an introduction to human-computer interaction for computer science majors. It places emphasis on understanding human behavior with interactive objects, knowing how to develop and evaluate interactive software using a human-

centered approach, and general knowledge of HCI design issues with multiple types of interactive software.

HC - Foundations of human computer interaction

Range of coverage time: 1-3 hours
Topics:

Motivation: Why care about people?
Contexts for HCI (tools, web hypermedia, communication)
Human-centered development and evaluation
Human performance models: perception, movement, and cognition
Human performance models: culture, communication, and organizations
Accommodating human diversity
Principles of good design and good designers; engineering tradeoffs
Introduction to usability testing

Learning objectives:

1. Discuss the reasons for human-centered software development.
2. Summarize the basic science of psychological and social interaction.
3. Differentiate between the role of hypotheses and experimental results vs. correlations.
4. Develop a conceptual vocabulary for analyzing human interaction with software: affordance, conceptual model, feedback, and so forth.
5. Distinguish between the different interpretations that the same icon, symbol, word, and color have among varying human cultures.
6. Identify ways to respect human diversity when interacting with a computer system.
7. Create and conduct a simple usability test for an existing software application.

HC - Building a simple graphical user interface

Range of coverage time: 1-2 hours
Topics:

Principles of graphical user interfaces (GUIs)
GUI toolkits

Learning objectives:

1. Identify several fundamental principles for effective GUI design.
2. Use a GUI toolkit to create a simple application that supports a graphical user interface.
3. Create two instances of the same GUI design; one based on fundamental design principles and the other ignoring these principles.
4. Conduct a simple usability test for each instance and compare the results.

HC - Graphical user-interface design

Range of coverage time: 0-4 hours

Topics:

Choosing interaction styles and interaction techniques
HCI aspects of common widgets
HCI aspects of screen design: layout, color, fonts, labeling
Handling human failure
Beyond simple screen design: visualization, representation, metaphor
Multi-modal interaction: graphics, sound, and haptics
3D interaction and virtual reality

Learning objectives:

1. Summarize common interaction styles.
2. Explain good design principles of each of the following: common widgets; sequenced screen presentations; simple error-trap dialog; a user manual.
3. Design, prototype, and evaluate a simple 2D GUI
4. Discuss the challenges that exist in moving from 2D to 3D interaction.

HC - HCI aspects of multimedia systems

Range of coverage time: 0-1 hours

Topics:

Categorization and architectures of information: hierarchies, hypermedia
Information retrieval and human performance
- Web search
- Usability of database query languages
- Graphics
- Sound

HCI design of multimedia information systems
Speech recognition and natural language processing
Information appliances and mobile computing

Learning objectives:

1. Discuss how information retrieval differs from transaction processing.
2. Explain how the organization of information supports retrieval.
3. Describe the major usability problems with database query languages.
4. Explain the current state of speech recognition technology in particular and natural language processing in general.
5. Design, prototype, and evaluate a simple Multimedia Information System.

2.7 Information Management (IM)

- **Information models and systems**
- **Database systems**
- **Database query languages**
- **Hypertext and hypermedia**

Information Management (IM) plays a critical role in almost all areas where computers are used. This area includes the capture, digitization, representation, organization, transformation, and presentation of information; algorithms for efficient and effective access and updating of stored information, data modeling and abstraction, and physical file storage techniques. It also encompasses information security, privacy, integrity, and protection in a shared environment. The student needs to be able to develop conceptual and physical data models, determine what IM methods and techniques are appropriate for a given problem, and be able to select and implement an appropriate IM solution that reflects all suitable constraints, including scalability and usability.

IM - Information models and systems

Range of coverage time: 0-3 hours
Topics:

History and motivation for information systems
Information storage and retrieval (IS&R)
Information management applications
Information capture and representation
Analysis and indexing
Search, retrieval, linking, navigation
Information privacy, integrity, security, and preservation
Scalability, efficiency, and effectiveness

Learning objectives:

1. Compare and contrast information with data and knowledge.
2. Summarize the evolution of information systems from early visions up through modern offerings, distinguishing their respective capabilities and future potential.
3. Critique/defend a small- to medium-size information application with regard to its satisfying real user information needs.
4. Describe several technical solutions to the problems related to information privacy, integrity, security, and preservation.
5. Explain measures of efficiency (throughput, response time) and effectiveness (recall, precision).
6. Describe approaches to ensure that information systems can scale from the individual to the global.

IM - Database systems

Range of coverage time: 0-6 hours
Topics:

History and motivation for database systems
Components of database systems
DBMS functions
Database architecture and data independence
Use of a database query language

Learning objectives:

1. Explain the characteristics that distinguish the database approach from the traditional approach of programming with data files.
2. Cite the basic goals, functions, models, components, applications, and social impact of database systems.
3. Describe the components of a database system and give examples of their use.
4. Identify major DBMS functions and describe their role in a database system.
5. Explain the concept of data independence and its importance in a database system.
6. Use a query language to elicit information from a database.

IM - Database query languages

Range of coverage time: 0-6 hours

Topics:

Overview of database languages
SQL (data definition, query formulation, update sublanguage, constraints, integrity)
Query optimization
QBE and 4th-generation environments
Embedding non-procedural queries in a procedural language
Introduction to Object Query Language

Learning objectives:

1. Create a relational database schema in SQL that incorporates key, entity integrity, and referential integrity constraints.
2. Demonstrate data definition in SQL and retrieving information from a database using the SQL **`SELECT`** statement.
3. Evaluate a set of query processing strategies and select the optimal strategy.
4. Create a non-procedural query by filling in templates of relations to construct an example of the desired query result.
5. Embed object-oriented queries into a stand-alone language such as C++ or Java (e.g., **`SELECT Col.Method() FROM Object`**).

IM - Hypertext and hypermedia

Range of coverage time: 0-3 hours

Topics:

Hypertext models (early history, web, Dexter, Amsterdam, HyTime)
Link services, engines, and (distributed) hypertext architectures
Nodes, composites, and anchors
Dimensions, units, locations, spans
Browsing, navigation, views, zooming
Automatic link generation
Presentation, transformations, synchronization
Authoring, reading, and annotation
Protocols and systems (including web, HTTP)

Learning objectives:

1. Summarize the evolution of hypertext and hypermedia models from early versions up through current offerings, distinguishing their respective capabilities and limitations.
2. Explain basic hypertext and hypermedia concepts.
3. Demonstrate a fundamental understanding of information presentation, transformation, and synchronization.
4. Compare and contrast hypermedia delivery based on protocols and systems used.
5. Design and implement web-enabled information retrieval applications using appropriate authoring tools.

2.8 Net-Centric Computing (NC)

- **Introduction to net-centric computing**
- **Communication and networking**
- **Network security**
- **The Web as an example of client-server computing**
- **Building web applications**
- **Network management**
- **Compression and decompression**
- **Multimedia data technologies**

Recent advances in computer and telecommunications networking, particularly those based on TCP/IP, have increased the importance of networking technologies in the computing discipline. Net-centric computing covers a range of sub-specialties including: computer communication network concepts and protocols, multimedia systems, Web standards and technologies, network security, wireless and mobile computing, and distributed systems.

Mastery of this subject area involves both theory and practice. Learning experiences that involve hands-on experimentation and analysis are strongly recommended as they

reinforce student understanding of concepts and their application to real-world problems. Laboratory experiments should involve data collection and synthesis, empirical modeling, protocol analysis at the source code level, network packet monitoring, software construction, and evaluation of alternative design models. All of these are important concepts that can best understood by laboratory experimentation.

NC - Introduction to net-centric computing

Range of coverage time: 0-5 hours
Topics:

Background and history of networking and the Internet
Network architectures
The range of specializations within net-centric computing
– Networks and protocols
– Networked multimedia systems
– Distributed computing
– Mobile and wireless computing

Learning objectives:

1. Discuss the evolution of early networks and the Internet.
2. Demonstrate a range of common networked applications including e-mail, telnet, ftp, newsgroups, and web browsers, on-line web courses and instant messaging..
3. Explain the hierarchical, layered structure of typical network architectures.
4. Describe emerging technologies in the net-centric computing area such as wireless computing and voice over IP.

NC - Communication and networking

Range of coverage time: 0-5 hours
Topics:

Network standards and standardization bodies
The ISO 7-layer reference model in general and its instantiation in TCP/IP
Circuit switching and packet switching
Streams and datagrams
Physical layer networking concepts (theoretical basis, transmission media, standards)
Data link layer concepts (framing, error control, flow control, protocols)
Internetworking and routing (routing algorithms, internetworking, congestion control)
Transport layer services (connection establishment, performance issues)

Learning objectives:

1. Discuss important network standards in their historical context.
2. Describe the first four layers of the ISO reference model.
3. Discuss the differences between circuit switching, message switching and packet switching along with the advantages and disadvantages of each.
4. Explain how a network can detect and correct transmission errors.

5. Illustrate the routing of a packet over the Internet.
6. Install a simple network with two clients and a single server using standard host-configuration software tools such as DHCP.

NC - Network security

Range of coverage time: 0-5 hours
Topics:

Fundamentals of cryptography
Secret-key algorithms
Public-key algorithms
Authentication protocols
Digital signatures
Examples

Learning objectives:

1. Discuss the fundamental ideas of public-key cryptography.
2. Distinguish between the use of secret and public-key algorithms.
3. Summarize common authentication protocols.
4. Generate and distribute a PGP key pair and use the PGP package to send an encrypted e-mail message.

NC - The Web as an example of client-server computing

Range of coverage time: 0-8 hours
Topics:

Web technologies
– Server-side programs
– Common gateway interface (CGI) programs
– Client-side scripts
– The applet concept
Characteristics of web servers
– Handling permissions
– File management
– Capabilities of common server architectures
Role of client computers
Nature of the client-server relationship
Web protocols
Support tools for web site creation and web management
Developing Internet information servers
Publishing information and applications

Learning objectives:

1. Explain the different roles and responsibilities of clients and servers for a range of possible applications.
2. Select a range of tools that will ensure an efficient approach to implementing a given client-server possibilities.
3. Design and build a simple interactive web-based application (e.g., a simple web form that collects information from the client and stores it in a file on the server).

NC - Building web applications

Range of coverage time: 0-9 hours

Topics:

Protocols at the application layer
Principles of web engineering
Database-driven web sites
Remote procedure calls (RPC)
Lightweight distributed objects
The role of middleware
Support tools
Security issues in distributed object systems
Enterprise-wide web-based applications

Learning objectives:

1. Illustrate how to build interactive client-server web applications of medium size using different types of Web technologies.
2. Demonstrate how to implement a database-driven web site, explaining the relevant technologies involved in each tier of the architecture and the accompanying performance tradeoffs.
3. Implement a distributed system using any two distributed object frameworks and compare them with regard to performance and security issues.
4. Discuss security issues in an enterprise-wide web-based application.

NC - Network management

Range of coverage time: 0-4 hours

Topics:

Overview of the issues of network management
Use of passwords and access control mechanisms
Domain names and name services
Issues for Internet service providers (ISPs)
Security issues and firewalls
Quality of service issues: performance, failure recovery

Learning objectives:

1. Explain the issues for network management arising from a range of security threats, including viruses, worms, Trojan horses, and denial-of-service attacks
2. Summarize the strengths and weaknesses associated with different approaches to security.
3. Develop a strategy for ensuring appropriate levels of security in a system designed for a particular purpose.
4. Implement a network firewall.

NC - Compression and decompression

Range of coverage time: 0-4 hours

Topics:

Analog and digital representations
Encoding and decoding algorithms
Lossless and lossy compression
Data compression: Huffman coding and the Ziv-Lempel algorithm
Audio compression and decompression
Image compression and decompression
Video compression and decompression
Performance issues: timing, compression factor, and suitability for real-time use

Learning objectives:

1. Summarize the basic characteristics of sampling and quantization for digital representation.
2. Select, giving reasons that are sensitive to the specific application and particular circumstances, the most appropriate compression techniques for text, audio, image, and video information.
3. Explain the asymmetric property of compression and decompression algorithms.
4. Illustrate the concept of run-length encoding.
5. Illustrate how a program like the UNIX compress utility, which uses Huffman coding and the Ziv-Lempel algorithm, would compress a typical text file.

NC - Multimedia data technologies

Range of coverage time: 0-7 hours

Topics:

Sound and audio, image and graphics, animation and video
Multimedia standards (audio, music, graphics, image, telephony, video, TV)
Capacity planning and performance issues
Input and output devices (scanners, digital camera, touch-screens, voice-activated)
MIDI keyboards, synthesizers
Storage standards (Magneto Optical disk, CD-ROM, DVD)
Multimedia servers and file systems

Tools to support multimedia development

Learning objectives:

1. Evaluate the potential of a computer system to host one of a range of possible multimedia applications, including an assessment of the requirements of multimedia systems on the underlying networking technology.
2. Describe the characteristics of a computer system (including identification of support tools and appropriate standards) that has to host the implementation of one of a range of possible multimedia applications.
3. Implement a multimedia application of modest size.

2.9 Operating Systems (OS)

- **Overview of operating systems**
- **Operating system principles**
- **Concurrency**
- **Scheduling and dispatch**
- **Memory management**
- **Device management**
- **Security and protection**
- **File systems**
- **Scripting**

An operating system defines an abstraction of hardware behavior with which programmers can control the hardware. It also manages resource sharing among the computer's users. Over the years, operating systems and their abstractions have become complex relative to typical application software. It is necessary to ensure that the student understands the extent of the use of an operating system before a detailed study of internal implementation algorithms and data structures. Therefore this content area addresses both the use of operating systems (externals) and their design and implementation (internals).

Many of the ideas involved in operating system use have wider applicability across the field of computer science, such as concurrent programming. Studying internal design has relevance in such diverse areas as dependable programming, algorithm design and implementation, modern device development, building virtual environments, caching material across the web, building secure and safe systems, network management, and many others.

OS - Overview of operating systems

Range of coverage time: 0-4 hours
Topics:

Role and purpose of the operating system
History of operating system development
Functionality of a typical operating system
Mechanisms to support client-server models, hand-held devices
Design issues (efficiency, robustness, flexibility, portability, security, compatibility)
Influences of security, networking, multimedia, windows

Learning objectives:

1. Explain the objectives and functions of modern operating systems.
2. Describe how operating systems historically have evolved from primitive batch systems to sophisticated multi-user systems.
3. Analyze the tradeoffs inherent in operating system design.
4. Describe the functions of a contemporary operating system with respect to convenience, efficiency, and the ability to evolve.
5. Discuss networked, client-server, distributed operating systems and how they differ from single user operating systems.
6. Identify potential threats to operating systems and the security features design to guard against them.
7. Describe how current issues such as open source software and the increased use of the Internet are influencing operating system design.

OS - Operating system principles

Range of coverage time: 0-1 hours
Topics:

Structuring methods (monolithic, layered, modular, micro-kernel models)
Abstractions, processes, and resources
Concepts of application program interfaces (APIs)
Applications needs and the evolution of hardware/software techniques
Device organization
Interrupts: methods and implementations
Concept of user/system state and protection, transition to kernel mode

Learning objectives:

1. Explain the concept of a logical layer.
2. Explain the benefits of building abstract layers in hierarchical fashion.
3. Recognize the need for APIs and middleware
4. Describe how computing resources are used by application software and managed by system software.

5. Compare system state and kernel mode to user-protected mode..
6. Discuss the advantages and disadvantages of using interrupt processing
7. Compare and contrast the various ways of structuring an operating system such as object oriented, modular, micro-kernel, and layered.
8. Explain the use of a device list and driver I/O queue.

OS - Concurrency

Range of coverage time: 0-3 hours
Topics:

States and state diagrams
Structures (ready list, process control blocks, and so forth)
Dispatching and context switching
The role of interrupts
Concurrent execution: advantages and disadvantages
The "mutual exclusion" problem and some solutions
Deadlock: causes, conditions, prevention
Models and mechanisms (semaphores, monitors, condition variables, rendezvous)
Producer-consumer problems and synchronization
Multiprocessor issues (spin-locks, reentrancy)

Learning objectives:

1. Describe the need for concurrency within the framework of an operating system.
2. Demonstrate the potential run-time problems arising from the concurrent operation of many (possibly a dynamic number of) tasks.
3. Summarize the range of mechanisms (at an operating system level) that can be employed to realize concurrent systems and describe the benefits of each.
4. Explain the different states that a task may pass through and the data structures needed to support the management of many tasks.
5. Summarize the necessary hardware support and the various software approaches to solving the problem of mutual exclusion in an operating system.
6. Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system.
7. Create state and transition diagrams.
8. Discuss the data structures, such as stack and queue, required to support concurrency.
9. Explain conditions that lead to deadlock, spin-locks, and reentrancy.

OS - Scheduling and dispatch

Range of coverage time: 0-5 hours
Topics:

Preemptive and non-preemptive scheduling
Schedulers and policies
Processes and threads
Deadlines and real-time issues

Learning objectives:

1. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems such as priority, performance comparison, and fair-share.
2. Describe relationships between scheduling algorithms and application domains.
3. Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O.
4. Describe the difference between processes and threads.
5. Compare and contrast both static and dynamic approaches to real-time scheduling.
6. Discuss the need for preemption and deadline scheduling.

OS - Memory management

Range of coverage time: 0-4 hours
Topics:

Review of physical memory and memory management hardware
Overlays, swapping, and partitions
Paging and segmentation
Placement and replacement policies
Working sets and thrashing
Caching

Learning objectives:

1. Introduce memory hierarchy and cost-performance tradeoffs.
2. Explain the meaning of virtual memory is and how hardware and software achieves it.
3. Examine the principles of virtual memory as applied to caching, paging, and segmentation.
4. Evaluate the tradeoffs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed.
5. Defend the different ways of allocating memory to tasks based on the relative merits of each.
6. Describe the reason for and use of cache memory.

7. Compare and contrast paging and segmentation techniques.
8. Analyze the various memory portioning techniques including overlays, swapping, and placement and replacement policies.

OS - Device management

Range of coverage time: 0-1 hours

Topics:

Characteristics of serial and parallel devices
Abstracting device differences
Buffering strategies
Direct memory access
Recovery from failures

Learning objectives:

1. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system.
2. Differentiate the mechanisms used in interfacing a range of devices (including hand-held devices, networks, multimedia) to a computer and explain the implications of these for the design of an operating system.
3. Implement a simple device driver for a range of possible devices.

OS - Security and protection

Range of coverage time: 0-2 hours

Topics:

Overview of system security
Policy/mechanism separation
Security methods and devices
Protection, access, and authentication
Models of protection
Memory protection
Encryption
Recovery management

Learning objectives:

1. Defend the need for protection and security, and the role of ethical considerations in computer use.
2. Summarize the features and limitations of an operating system used to provide protection and security.
3. Compare and contrast current methods for implementing security.

OS - File systems

Range of coverage time: 0-3 hours

Topics:

Files: data, metadata, operations, organization, buffering, sequential, nonsequential
Directories: contents and structure
File systems: partitioning, mount/unmount, and virtual file systems
Standard implementation techniques
Memory-mapped files
Special-purpose file systems
Naming, searching, access, backups

Learning objectives:

1. Summarize the full range of considerations that support file systems.
2. Compare and contrast different approaches to file organization, including the strengths and weaknesses of each.

OS - Scripting

Range of coverage time: 0-3 hours

Topics:

Scripting and the role of scripting languages
Basic system commands
Creating scripts, parameter passing
Executing a script
Influences of scripting on programming

Learning objectives:

1. Summarize a typical set of system commands provided by an operating system.
2. Demonstrate the typical functionality of a scripting language, and interpret the implications for programming.
3. Demonstrate the mechanisms for implementing scripts and the role of scripts on system implementation and integration.
4. Implement a simple script that exhibits parameter passing.

2.10 Programming Fundamentals (PF)

- **Fundamental programming constructs**
- **Algorithms and problem solving**
- **Fundamental data structures**
- **Recursion**
- **Event-driven programming**

This knowledge area consists of those skills and concepts that are essential to programming practice independent of the underlying paradigm. As a result, this area includes units on fundamental programming concepts, basic data structures, and algorithmic processes. These units, however, by no means cover the full range of programming knowledge that a computer science undergraduate must know. Many of the other areas—most notably Programming Languages (PL) and Software Engineering (SE)—also contain programming-related units that are part of the undergraduate core. In most cases, these units could apply to either Programming Fundamentals or the more advanced area.

PF - Fundamental programming constructs

Range of coverage time: 9-13 hours
Topics:

Basic 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

Learning objectives:

1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs.
2. Explain the use of each data type and how each is stored in memory.
3. Modify and expand short programs using control structures and functions.
4. Design, implement, test and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.
5. Choose appropriate selection and iteration constructs for a given programming task.
6. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
7. Describe parameters passing between functions.

PF - Algorithms and problem solving

Range of coverage time: 6-9 hours
Topics:

Problem-solving strategies
The role of algorithms in the problem-solving process
Implementation strategies for algorithms
Debugging strategies
The concept and properties of algorithms

Learning objectives:

1. Discuss why algorithms are useful in problem solving with a programming language.
2. List the recommended steps in problem solving.
3. Create algorithms for solving simple problems.
4. Use pseudocode or a programming language to implement, test, and debug algorithms for problem solving.
5. Discuss what makes a good algorithm.
6. Analyze an algorithm's correctness and efficiency.

PF - Fundamental data structures

Range of coverage time: 11-19 hours
Topics:

Primitive types
Arrays
Records
Strings and string processing
Data representation in memory
Static, stack, and heap allocation
Runtime storage management
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

Learning objectives:

1. Define a data structure and an Abstract Data Type (ADT) and distinguish between built-in and user-defined data structures.
2. Discuss the representation and use of primitive data types and built-in data structures.
3. Describe common applications for each data structure covered.
4. Describe ADTs at a logical or abstract level and discuss how each works.

5. Implement the user-defined data structures in a high-level language.
6. Compare alternative implementations of data structures with respect to performance.
7. Write and execute a program for testing a data structure implementation.
8. Compare and contrast dynamic and static data structure implementations.
9. Choose the appropriate data structure for modeling a given problem.

PF - Recursion

Range of coverage time: 3-8 hours
Topics:

The concept of recursion
Recursive mathematical functions
Simple recursive procedures
Divide-and-conquer strategies
Recursive backtracking
Implementation of recursion

Learning objectives:

1. Describe and exemplify the concept of recursion.
2. Verify correctness of a recursive routine by identifying the base case and the general case.
3. Compare iterative and recursive solutions for elementary problems such as factorial.
4. Define the divide-and-conquer approach.
5. Implement, test, and debug simple recursive functions.
6. Describe how recursion can be implemented using a stack.
7. Discuss problems for which backtracking is an appropriate solution.
8. Determine when a recursive solution is appropriate for a problem.

PF - Event-driven programming

Range of coverage time: 1-3 hours
Topics:

Event-handling methods
Event propagation
Exception handling

Learning objectives:

1. Explain the difference between event-driven programming and command-line programming.
2. Identify programming languages that support event-driven programming and those that do not.
3. Design, code, test, and debug simple event-driven programs that respond to user events.
4. Develop code that responds to exception conditions raised during execution.

2.11 Programming Languages (PL)

- **Overview of programming languages**
- **Virtual machines**
- **Introduction to language translation**
- **Declarations and types**
- **Abstraction mechanisms**
- **Object-oriented programming**
- **Functional programming**

A programming language is a programmer's principal interface with the computer. More than just knowing how to program in a single language, programmers need to understand the different styles of programming promoted by different languages. In their professional life, they will be working with many different languages and styles at once, and will encounter many different languages over the course of their careers.

Understanding the variety of programming languages and the design tradeoffs between the different programming paradigms makes it much easier to master new languages quickly. Understanding the pragmatic aspects of programming languages also requires a basic knowledge of programming language translation and runtime features such as storage allocation.

PL - Overview of programming languages

Range of coverage time: 2 hours
Topics:

History of programming languages
Brief survey of programming paradigms
– Procedural languages
– Object-oriented languages
– Functional languages
– Declarative, non-algorithmic languages
– Scripting languages
The effects of scale on programming methodology

Learning objectives:

1. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today.
2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit.
3. Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
4. Define issues concerning programming-in-the-small versus programming-in-the-large.

PL - Virtual machines

Range of coverage time: 1 hour
Topics:

The concept of a virtual machine
Hierarchy of virtual machines
Intermediate languages
Security issues arising from running code on an alien machine

Learning objectives:

1. Describe the importance and power of abstraction in the context of virtual machines.
2. Explain the benefits of intermediate languages in the compilation process.
3. Evaluate the tradeoffs in performance vs. portability.
4. Explain how executable programs can breach computer system security by accessing disk files and memory.

PL - Introduction to language translation

Range of coverage time: 0-4 hours
Topics:

Comparison of interpreters and compilers
Language translation phases (lexical analysis, parsing, code generation, optimization)
Machine-dependent and machine-independent aspects of translation

Learning objectives:

1. Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
2. Describe the phases of program translation from source code to executable code and the files produced by these phases.
3. Explain the differences between machine-dependent and machine-independent translation.

PL - Declarations and types

Range of coverage time: 3 hours
Topics:

The conception of types as a set of values together with a set of operations
Declaration models (binding, visibility, scope, and lifetime)
Overview of type checking
Garbage collection

Learning objectives:

1. Explain the value of declaration models, especially with respect to programming-in-the-large.
2. Identify the properties of a variable [object] such as its associated address, value, scope, persistence, and size.
3. Discuss type incompatibility.
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management.
5. Defend the importance of types and type checking in providing abstraction and safety.
6. Evaluate tradeoffs in lifetime management (reference counting vs. garbage collection).

PL - Abstraction mechanisms

Range of coverage time: 3 hours
Topics:

Procedures, functions, and iterators as abstraction mechanisms
Parameterization mechanisms (reference vs. value)
Activation records and storage management
Type parameters and parameterized types
Modules in programming languages

Learning objectives:

1. Explain how abstraction mechanisms support the creation of reusable software components.
2. Demonstrate the difference between call-by-value and call-by-reference parameter passing.
3. Defend the importance of abstractions, especially with respect to programming-in-the-large.
4. Describe how the computer system uses activation records to manage program modules and their data.

PL - Object-oriented programming

Range of coverage time: 12-14 hours
Topics:

Object-oriented design
Encapsulation and information hiding
Separation of specification and implementation
Classes and subclasses
Inheritance (overriding, dynamic dispatch)
Polymorphism (subtype polymorphism vs. inheritance)
Class hierarchies

Collection classes and iteration protocols
Internal representations of objects and method tables

Learning objectives:

1. Justify the philosophy of object-oriented design and the concepts of encapsulation, abstraction, inheritance, and polymorphism.
2. Design, implement, test, and debug simple programs in an object-oriented programming language.
3. Defend the benefits of separating class specification from class implementation
4. Describe how the class mechanism supports encapsulation and information hiding.
5. Design, implement, and test the implementation of "is-a" relationships among objects using a class hierarchy and inheritance.
6. Implement polymorphism using virtual functions.
7. Compare and contrast the notion of [static] overloading v. [run-time] overriding functions.
8. Describe the [different levels] use of selective inheritance.
9. Demonstrate the use of generic program components that use types as parameters.
10. Explain the relationship between the static structure of the class and the dynamic structure of the instances [objects] of the class.
11. Describe how iterators access the elements of a container.

PL - Functional programming

Range of coverage time: 0-1 hours
Topics:

Overview and motivation of functional languages
Recursion over lists, natural numbers, trees, and other recursively-defined data
Pragmatics (debugging by divide and conquer; persistency of data structures)
Amortized efficiency for functional data structures
Closures and uses of functions as data (infinite sets, streams)

Learning objectives:

1. Outline the strengths and weaknesses of the functional programming paradigm.
2. Design, code, test, and debug programs using the functional paradigm.
3. Explain the use of functions as data, including the concept of closures.

2.12 Software Engineering (SE)

- **Software design**
- **Using APIs**
- **Software tools and environments**
- **Software processes**
- **Software requirements and specifications**
- **Software validation**
- **Software evolution**
- **Software project management**

Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers. Software engineering is applicable to small, medium, and large-scale systems. It encompasses all phases of the life cycle of a software system. The life cycle includes requirement analysis and specification, design, construction, testing, and operation and maintenance.

Software engineering employs engineering methods, processes, techniques, and measurement. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. Software development, which can involve an individual developer or a team of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment.

The elements of software engineering are applicable to the development of software in any computing application domain where professionalism, quality, schedule, and cost are important in producing a software system.

SE - Software design

Range of coverage time: 3-7 hours
Topics:

Fundamental design concepts and principles
Design patterns
Software architecture
Structured design
Object-oriented analysis and design
Component-level design
Design for reuse

Learning objectives:

1. Discuss the properties of good software design.
2. Compare and contrast object-oriented analysis and design with structured analysis and design.
3. Evaluate the quality of multiple software designs based on key design principles and concepts.
4. Select and apply appropriate design patterns in the construction of a software application.
5. Create and specify the software design for a medium-size software product using a software requirement specification and the Unified Modeling Language (UML).
6. Conduct a software design review using appropriate guidelines.
7. Evaluate a software design at the component level and evaluate for reuse.

SE - Using APIs

Range of coverage time: 1-2 hours
Topics:

API programming
Class browsers and related tools
Programming by example
Debugging in the API environment
Introduction to component-based computing

Learning objectives:

1. Explain the value of application programming interfaces (APIs) in software development.
2. Explain the difference between sequential and event-driven programming.
3. Discuss benchmark programs using APIs.
4. Use class browsers and related tools during the development of applications using APIs.
5. Design, implement, test, and debug programs that use large-scale API packages.

SE - Software tools and environments

Range of coverage time: 0-2 hours

Topics:

Programming environments
Requirements analysis and design modeling tools
Testing tools
Configuration management tools
Tool integration mechanisms

Learning objectives:

1. Select, with justification, an appropriate set of CASE tools to support the software development of a range of software products.
2. Analyze and evaluate a set of tools in a given area of software development (e.g., management, modeling, or testing).
3. Demonstrate the use of a range of software tools in support of the development of a software product of medium size.

SE - Software processes

Range of coverage time: 0-1 hours
Topics:

Software life cycle and process models
Process assessment models
Software process metrics

Learning objectives:

1. Explain the software life cycle, define its phases, and describe the deliverables that are produced.
2. Select, with justification the software development models most appropriate for the development and maintenance of diverse software products.
3. Explain the role of process maturity models.
4. Analyze the software engineering process using standard metrics.
5. Review ISO 9000 standards
6. Compare the traditional waterfall model to the incremental model, the object-oriented model, and other select models.

SE - Software requirements and specifications

Range of coverage time: 0-1 hours
Topics:

Requirements elicitation
Requirements analysis modeling techniques
Functional and nonfunctional requirements
Prototyping
Basic concepts of formal specification techniques

Learning objectives:

1. Apply key elements and common methods for elicitation and analysis to produce a set of software requirements for a medium-sized software system.
2. Discuss the challenges of maintaining legacy software.

3. Use a common, non-formal method to model and specify (in the form of a requirements specification document) the requirements for a medium-size software system (e.g., structured analysis or object-oriented-analysis).
4. Conduct a review of a software requirements document using best practices to determine the quality of the document.
5. Translate into natural language a software requirements specification written in a commonly used formal specification language.
6. Demonstrate a commonly used prototyping tool.

SE - Software validation

Range of coverage time: 0-2 hours
Topics:

Validation planning
Testing fundamentals, including test plan creation and test case generation
Black-box and white-box testing techniques
Unit, integration, validation, and system testing
Object-oriented testing
Inspections

Learning objectives:

1. Distinguish between program validation and verification.
2. Distinguish between and implement the different types and levels of testing (unit, integration, systems, and acceptance) for medium-size software products.
3. Create, evaluate, and implement a test plan for a medium-size code segment.
4. Undertake, as part of a team activity, an inspection of a medium-size code segment.
5. Describe the role that tools can play in the validation of software.
6. Discuss the issues involving the testing of object-oriented software.

SE - Software evolution

Range of coverage time: 0-1 hours
Topics:

Software maintenance
Characteristics of maintainable software
Reengineering
Legacy systems
Software reuse

Learning objectives:

1. Identify the principal issues associated with software evolution and explain their impact on the software life cycle.
2. Outline the process of regression testing and its role in release management.

3. Estimate the impact of a change request to an existing product of medium size.
4. Develop a plan for re-engineering a medium-sized product in response to a change request.
5. Discuss the advantages and disadvantages of software reuse.
6. Demonstrate the ability to exploit opportunities for software reuse in a variety of contexts.

SE - Software project management

Range of coverage time: 0-1 hours
Topics:

Team management
- Team processes
- Team organization and decision-making
- Roles and responsibilities in a software team
- Role identification and assignment
- Project tracking
- Team problem resolution

Project scheduling
Software measurement and estimation techniques
Risk analysis
Software quality assurance
Software configuration management
Project management tools

Learning objectives:

1. Demonstrate through involvement in a team project the central elements of team building and team management.
2. Prepare a project plan for a software project that includes estimates of size and effort, a schedule, resource allocation, configuration control, change management, and project risk identification and management.
3. Compare and contrast the different methods and techniques used to assure the quality of a software product.

2.13 Social and Professional Issues (SP)

- **History of computing**
- **Social context of computing**
- **Methods and tools of analysis**
- **Professional and ethical responsibilities**
- **Risks and liabilities of computer-based systems**
- **Intellectual property**

SP - History of computing

Range of coverage time: 1 hour
Topics:

Prehistory—the world before 1946
History of computer hardware, software, networking
Pioneers of computing

Learning objectives:

1. List the contributions of several pioneers in the computing field.
2. Compare daily life before and after the advent of computing.
3. Identify significant continuing trends in the history of the computing field and their impact on society.

SP - Social context of computing

Range of coverage time: 1 hour
Topics:

Introduction to the social implications of computing
Social implications of networked communication
Growth of, control of, and access to the Internet
Gender-related issues
International issues

Learning objectives:

1. Interpret the social context of a particular implementation.
2. Identify assumptions and values embedded in a particular design.
3. Evaluate a particular implementation using empirical data.
4. Describe positive and negative ways in which computing alters the modes of interaction between people.
5. Explain why computing/network access is restricted in some countries.

SP - Methods and tools of analysis

Range of coverage time: 1 hour
Topics:

Making and evaluating ethical arguments
Identifying and evaluating ethical choices
Understanding the social context of design
Identifying assumptions and values

Learning objectives:

1. Analyze an argument to identify premises and conclusion.
2. Illustrate the use of example, analogy, and counter-analogy in ethical argument.
3. Detect use of basic logical fallacies in an argument.
4. Identify stakeholders in an issue and our obligations to them.
5. Articulate the ethical tradeoffs in a technical decision.

SP - Professional and ethical responsibilities

Range of coverage time: 0-1 hours
Topics:

Community values and the laws by which we live
The nature of professionalism
Various forms of professional credentialing and the advantages and disadvantages
The role of the professional in public policy
Maintaining awareness of consequences
Ethical dissent and whistle-blowing
Codes of ethics, conduct, and practice (IEEE, ACM, SE, AITP, and so forth)
Dealing with harassment and discrimination
"Acceptable use" policies for computing in the workplace
Computer crime

Learning objectives:

1. Identify progressive stages in a whistle-blowing incident.
2. Specify the strengths and weaknesses of relevant professional codes as expressions of professionalism and guides to decision-making.
3. Identify ethical issues that arise in software development and determine how to address them technically and ethically.
4. Develop a computer use policy with enforcement measures.
5. Analyze a global computing issue like Y2K as a case study, observing the role of professionals and government officials in managing the problem.
6. Evaluate the professional codes of ethics from the ACM, the IEEE Computer Society, and other organizations.

SP - Risks and liabilities of computer-based systems

Range of coverage time: 0-1 hours
Topics:

Historical examples of software risks (such as the Therac-25 case)
Implications of software complexity
Risk assessment and management
Computer crime

Learning objectives:

1. Explain, using case studies, the limitations of testing as a means to ensure correctness.
2. Describe the differences between correctness, reliability, and safety.
3. Recognize unwarranted assumptions that errors are statistically independent.
4. Discuss the potential for hidden problems in reuse of existing components.
5. Recognize the risks associated with various computer crimes.

SP - Intellectual property

Range of coverage time: 0-1 hours
Topics:

Foundations of intellectual property
Copyrights, patents, and trade secrets
Software piracy
Software patents
Transnational issues concerning intellectual property

Learning objectives:

1. Distinguish among patent, copyright, and trade secret protection including statue of limitation.
2. Discuss the legal background of copyright in national and international law.
3. Explain how patent and copyright laws may vary from country to country.
4. Outline the historical development of software patents.
5. Discuss the consequences of software piracy on software developers and the role of enforcement organizations such as the Software Publishers Association (SPA) relative to perpetrators.

Chapter 3. Curricula

3.1 Curriculum Overview

For each implementation strategy, this chapter provides a detailed package of courses that includes all of the recommended learning objectives for the particular implementation. In addition, each implementation should include courses to satisfy the college's general education requirements; related program electives; and sufficient other electives for the requisite credit hour total needed for an associate-degree program. The combination of general education and elective courses should also be chosen to provide the auxiliary skills specified in Chapter One of this report.

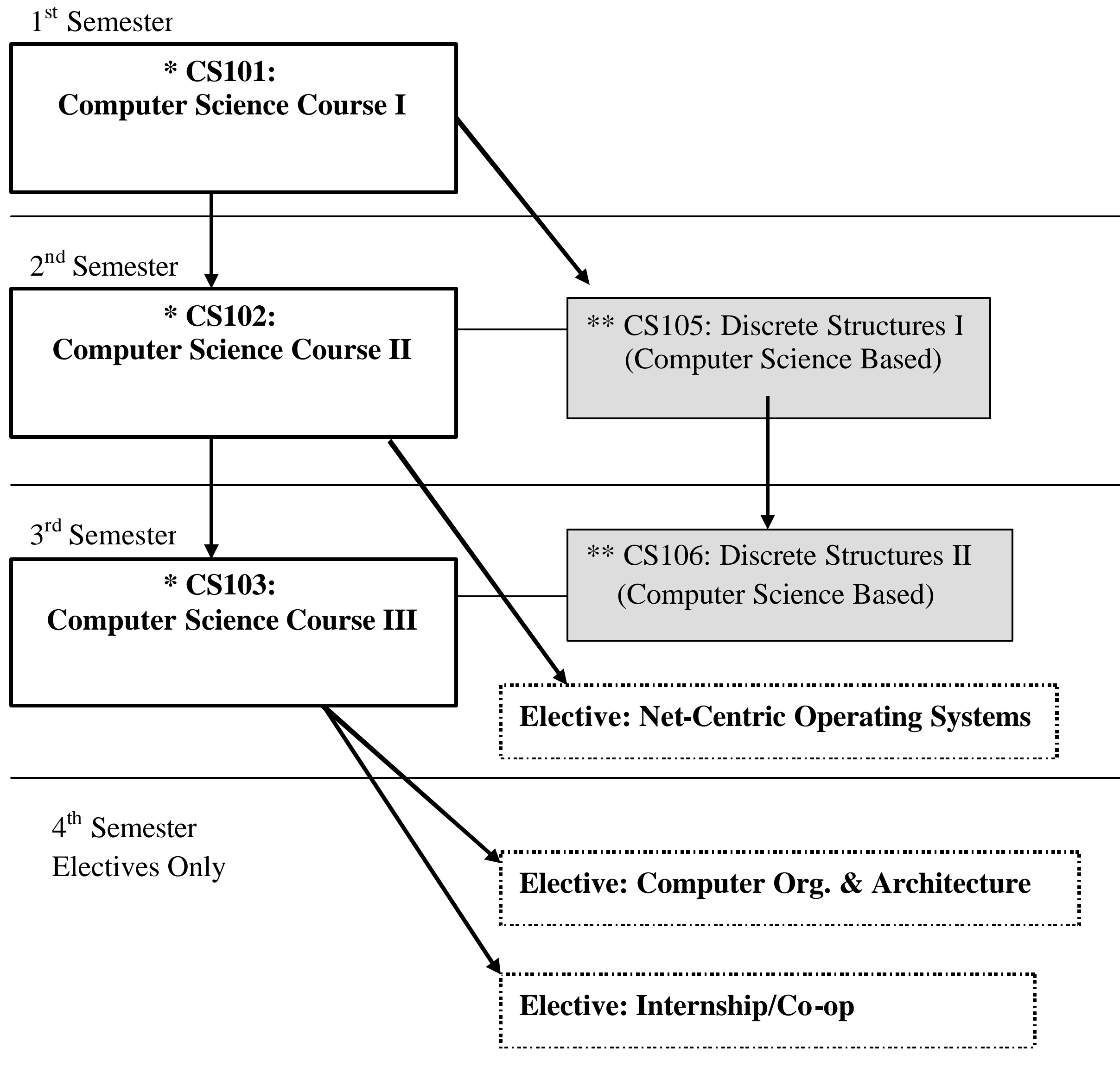
Each three-course sequence described in each of the three approaches in this report is fundamentally equivalent to the corresponding sequence in the IEEE-CS/ACM *CC2001 CS Report*, and each is designed to be at least equivalent to the corresponding two-course sequence in the *CC2001 CS Report*. A course-by-course comparison between the two reports will reveal differences that generally reflect additional time and preparatory material provided in the two-year college setting. Students completing any of the three-course sequences described herein should be well prepared for further work in computer science.

The following table and chart illustrate the sequencing for the required and elective computing and math courses in an associate-degree program in computer science. This course sequencing applies irrespective of the chosen implementation strategy. This course sequence assumes the student has prerequisite skills in computer fluency to enter the first computer science course (CS 101), and the fundamental mathematics necessary to enter into the first discrete mathematics course (CS 105).

Table 3.1: Possible Course Sequencing by Semester

FIRST SEMESTER	**SECOND SEMESTER**
Req'd: CS 101: Comp. Sci. Course I	Req'd: CS 102: Comp. Sci. Course II Req'd: CS 105: Discrete Structures I Elective: Intro. to Web-Centric Computing or Hardware Fundamentals
THIRD SEMESTER	**FOURTH SEMESTER**
Req'd: CS 103: Comp. Sci. Course III Req'd: CS 106: Discrete Structures II Elective: Introduction to Statistics Elective: Net-Centric Operating Systems	Elective: Computer Organization & Architecture. Elective: Hardware Fundamentals or Intro. to Web-Centric Computing Elective: Internship/Co-op

Chart 3.1 Course Sequencing and Prerequisite Relationships



Electives with no Computing Prerequisites:

Elective: Hardware Fundamentals

Elective: Intro Web-Centric Computing

Elective: Introduction to Statistics

* Required Computer Science
** Required Mathematics

3.2 The Objects-first Implementation Strategy

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.

Prerequisites: No programming or computer science experience is required; mathematical preparation sufficient to qualify for precalculus at the college level.

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

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental programming constructs	9
	Algorithms and problem-solving	4
	Fundamental data structures	3
AL	Fundamental computing algorithms	1
	Basic computability	1
PL	Overview of programming languages	2
	Declarations and types	2
	Object-oriented programming	7
	Functional programming	1
AR	Machine level representation of data	2
SE	Software tools and environments	1
	Software validation	1
SP	History of computing	1
	Social context of computing	1
	Professional and ethical responsibilities	1
	Risks and liabilities of computer-based systems	1
Other	Topics of local interest	2
	TOTAL:	**40**

Notes:

This course represents the first semester of an objects-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101O-102O-103O.

Although this is the first course of the educational process for a computer science student, it is reasonable to expect that a student will have at least some exposure to computers before taking this course. The prepared student will have experience with email, World Wide Web use, and basic word processing. Note that this course will not necessarily be taken during a student's first semester in college. Any remedial work (generally identified as "developmental studies" or "learning support" coursework) in mathematics or language arts should be completed before the student is allowed to begin this course sequence.

What differentiates this course sequence from the imperative-first implementation in CS101I-102I 103I is the early emphasis on objects. The discussion of classes, subclasses, and inheritance typically precedes even such basic concepts as conditional and iterative control statements.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

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.

Prerequisite: CS 101o
Corequisite: CS 105

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

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental data structures	3
	Recursion	2
	Event-driven programming	3
AL	Basic algorithmic analysis	1
	Fundamental computing algorithms	2
PL	Virtual machines	1
	Declarations and types	1
	Abstraction mechanisms	3
	Object-oriented programming	7
AR	Machine level representation of data	1
HC	Foundations of human-computer interaction	1
	Building a simple graphical user interface	1
GV	Fundamental techniques in graphics	2
	Graphic systems	1
SE	Software design	3
	Using APIs	2
	Software tools and environments	1
	Software requirements and specifications	1
	Software validation	1
	Software evolution	1
Other	Topics of local interest	2
	TOTAL:	**40**

Notes:

This course represents the second semester of an objects-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101O-102O-103O.

This second computer science course should be taken in parallel with the first discrete mathematics course, CS 105, to ensure that the appropriate mathematical foundations of computer science are covered along with the applications of those topics.

What differentiates this course sequence from the imperative-first implementation in CS101I-102I 103I is the early emphasis on objects. The discussion of classes, subclasses, and inheritance typically precedes even such basic concepts as conditional and iterative control statements.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered “lecture” hours and typically two (2) hours of lab equates to one (1) hour of lecture.

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.

Prerequisite: CS 102o
Corequisite: CS 106

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

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Algorithms and problem-solving	3
	Fundamental data structures	11
	Recursion	6
AL	Basic algorithmic analysis	3
	Algorithmic strategies	6
	Fundamental computing algorithms	5
SE	Software design	3
	Software project management	1
Other	Topics of local interest	2
	TOTAL:	**40**

Notes:

This course represents the third and final semester of an objects-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101O-102O-103O.

This third computer science course should be taken in parallel with the second discrete mathematics course, CS 106, to ensure that the appropriate mathematical foundations of computer science are covered along with the applications of those topics.

What differentiates this course sequence from the imperative-first implementation in CS101I-102I 103I is the early emphasis on objects. The discussion of classes, subclasses, and inheritance typically precedes even such basic concepts as conditional and iterative control statements.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

3.3 The Imperative-first Implementation Strategy

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.

Prerequisites: No programming or computer science experience is required; mathematical preparation sufficient to qualify for precalculus at the college level.

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.

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental programming constructs	10
	Algorithms and problem-solving	3
	Fundamental data structures	2
PL	Introduction to language translation	1
	Declarations and types	3
	Abstraction mechanisms	3
AR	Machine level representation of data	1
	Assembly level machine organization	2
OS	Overview of operating systems	1
NC	Introduction to net-centric computing	1
HC	Foundations of human-computer interaction	1
GV	Fundamental techniques in graphics	1
SE	Software design	3
	Software tools and environments	2
	Software processes	1
SP	History of computing	1
	Social context of computing	1
	Professional and ethical responsibilities	1
	Intellectual property	1
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the first semester of an imperative-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101I-102I-103I.

Although this is the first course of the educational process for a computer science student, it is reasonable to expect that a student will have at least some exposure to computers before taking this course. The prepared student will have experience with email, World Wide Web use, and basic word processing. Note that this course will not necessarily be taken during a student's first semester in college. Any remedial work (generally identified as "developmental studies" or "learning support" coursework) in mathematics or language arts should be completed before the student is allowed to begin this course sequence.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

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.

Prerequisite: CS 101I
Corequisite: CS 105

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

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental programming constructs	3
	Algorithms and problem-solving	6
	Fundamental data structures	5
	Event-driven programming	1
AL	Fundamental computing algorithms	3
PL	Overview of programming languages	1
	Virtual machines	1
	Introduction to language translation	1
	Object-oriented programming	6
AR	Machine level representation of data	2
HC	Foundations of human computer interaction	1
	Building a simple graphical user interface	2
IM	Database systems	1
SE	Software design	1
	Using APIs	2
	Software requirements and specifications	1
	Software validation	1
	Software evolution	1
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the second semester of an imperative-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101I-102I-103I.

This second computer science course should be taken in parallel with the first discrete mathematics course, CS 105, to ensure that the appropriate mathematical foundations of computer science are covered along with the applications of those topics.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

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.

Prerequisites: CS 102I
Corequisite: CS 106

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

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental data structures	12
	Recursion	5
AL	Basic algorithmic analysis	2
	Algorithmic strategies	3
	Fundamental computing algorithms	5
	Basic computability	1
PL	Overview of programming languages	1
	Object-oriented programming	8
SE	Software validation	1
	Software project management	1
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the third and final semester of an imperative-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101I-102I-103I.

This third computer science course should be taken in parallel with the second discrete mathematics course, CS 106, to ensure that the appropriate mathematical foundations of computer science are covered along with the applications of those topics.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered “lecture” hours and typically two (2) hours of lab equates to one (1) hour of lecture.

3.4 The Breadth-first Implementation Strategy

The Breadth-first approach covers a wide expanse of computer science topics in a three-course sequence, and emphasizes the mastery, selection and use of appropriate programming paradigms in a problem-solving context.

CS101B: Computing Sciences I

This course presents a broad overview of computer science that integrates programming with hardware fundamentals, algorithms, and computability. Topics include: Problem-solving strategies, study of algorithms, programming paradigms, and the social and historical context of computing.

Prerequisite: No programming or computer science experience is required; mathematical preparation sufficient to qualify for precalculus at the college level.

Syllabus:

- Computing overview
  - Layers of computing
    - Hardware
    - Machine level software
    - High level languages
    - Application software
  - History of Computing
    - Prehistory before 1946
    - History of computers hardware, software and networking
    - Pioneers of computing
    - History of programming languages
- Basic syntax and semantics of a higher level language
- Machine level representation
  - Bits, bytes and words
  - Numeric data representation and number bases
- Variables, types, expressions, and assignments
- Simple input and output
- Application programming interfaces (APIs) in software development
- Problem solving strategies
- Role of algorithms and the problem solving process
- Concepts and properties of algorithms
- Primitive data types
- Conditional and iterative control structures
- Implementation strategies for algorithms
- Debugging strategies
- Predicate logic and truth tables
- Brief survey of programming paradigms in relationship to programming languages
  - Procedural paradigm
  - Object oriented paradigm

- Functional paradigm
- Strings and string processing
- Event handling methods
- General principles and goals of language design
- Functions and parameter passing
- History of the Internet
- Basic reasons for human-centered development
- Social concepts and implications of computing network communications
- Foundations of intellectual property:
  - Copyright
  - Patents
  - Trade secrets
- Software Piracy
- Computer Crime
  - History
  - Viruses, worm, and Trojan horses
  - Prevention

Knowledge Area	***Units Covered***	***Suggested Hours***
PF	Fundamental programming concepts	9
	Algorithms and problem solving	6
	Fundamental data structures	3
	Event driven programming	3
SE	Software design	1
	Using API's introduction	1
PL	Overview of programming languages	2
	Declarations and types	3
	Abstraction mechanisms	3
AR	Digital Logic and digital systems	1
	Machine level representation of data	1
NC	Introduction to net-centric computing	2
HC	Foundations of human-computer interaction	1
SP	History of computing	1
	Social context of computing	1
	Intellectual property	1
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the first semester of a breadth-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental,

making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101B-102B-103B.

Although this is the first course of the educational process for a computer science student, it is reasonable to expect that a student will have at least some exposure to computers before taking this course. The prepared student will have experience with email, World Wide Web use, and basic word processing. Note that this course will not necessarily be taken during a student's first semester in college. Students should complete any remedial work (generally identified as "developmental studies" or "learning support" coursework) in mathematics or language arts before students can begin this course sequence.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

CS 102B: Computing Sciences II

Within this course structured data types are introduced into the curriculum. Arrays and records are described both as programming constructs and a method for the presentation of data organization in memory. Also, numeric representation in memory is covered. This course forms a foundation for formal object design along with a discussion of object-oriented theory. Topics include: Basic structured data types, object-oriented programming concepts, programming skills in the context of operating systems, information management, and client-server computing environments.

Prerequisite: CS 101B
Corequisite: CS 105

Syllabus:

- Object oriented design and the concepts of encapsulation, subclassing, inheritance, and polymorphism
- Design, code, test, and debug simple programs in an object oriented programming language
- Appropriate design patterns in the construction of object-oriented applications
- Arrays, records, references, and their data representation in memory
- Importance and power of abstraction in the context of virtual machines
- Benefits of intermediate languages in the compilation process
- Differences between compiled and interpreted execution models, outlining the relative merits of each
- Phases involved in language translation
- Role and purpose of operating systems
- History of operating system development
- Different roles and responsibilities of clients and servers in a web based environment
- Tools for use in a web-based environment
- Provide principles of good design of the human interface

- Design and construction of simple interactive Web based applications
- History of Information and database systems
- Information systems applications
- Privacy, integrity, and security of system applications
- Components of database systems
- Object query languages (e.g. SQL) for eliciting information from a database
- Object-oriented queries in stand-alone languages (e.g. C++ or Java)
- Concepts of records, record types, and files; different techniques for placing file records on disk
- Legal and ethical basis for computer software
- Internal representation of numeric data, characters, strings, records, and arrays
- Conversion of numerical data from one format to another

Knowledge Area	***Units Covered***	***Suggested Hours***
PF	Fundamental data structures	4
PL	Object oriented programming Virtual machines Introduction to language translation	12 1 2
AR	Machine level representation of data	2
OS	Overview of operation systems	2
NC	Web as client server computing	3
HC	Foundations of human-computer interaction	2
IM	Information models and systems Database systems Database query languages	3 3 3
SE	Software Design	2
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the second semester of a breadth-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101B-102B-103B.

Students should take this course in parallel with the first discrete mathematics course, CS 105, for majors in the computing sciences to ensure coverage of the appropriate mathematical foundations of computer science along with the applications of those topics.

There are two important considerations in the design of a breadth-first introduction to computer science. The first is to treat discrete mathematics not as a separate and unrelated subject, but as a fully integrated component of the course. By doing so, students will better understand and appreciate the importance of discrete mathematics to our discipline. For example, Boolean logic could be introduced during a discussion of programming language operators, counting methods could be presented during a discussion of the efficiency of iterative algorithms, while recurrence relations are a natural way to study the performance of recursive algorithms. The goal is for students to be introduced to mathematical concepts within the context of their use in solving important computing problems.

The second point is that the many disparate topics typically found in a breadth-first approach must be tied together into an integrated whole. Students must not see the course as a collection of interesting but unrelated topics in a "if this is Tuesday it must be computer organization" style. They should instead develop an appreciation for the important relationships among the major subfields of computer science. This goal can be achieved by demonstrating how each of the course topics utilizes earlier ideas and builds on them to produce newer and more powerful abstractions. This type of "spiral" implementation, which reinforces, emphasizes, and builds on previous concepts, is an important aspect to the success of a breadth-first approach.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

CS 103B: Computing Sciences III

The primary programming skills developed in this course are pointers and the concept of recursion. A limited discussion of linked lists may be necessary to properly anchor this material. Formal algorithm analysis fundamentals are also appropriate for introduction at this point, though some informal discussions of efficiency have probably occurred in previous discussions of algorithm development. The student should be exposed to time efficiency in the context of big O notation. Topics include: Advanced high-level language programming, basic assembly language skills, computer architecture, network-centered computing, and artificial intelligence.

Prerequisite: CS 102B
Corequisite: CS 106

Syllabus:

- Recursion, including implementation and debugging of simple recursive functions and procedures
- Pointers
- Big O notation
- Time complexity of simple algorithms

- Fundamental digital logic building blocks (logic gates, flip-flops, counters, registers, PLA)
- Register transfer notation
- Basic organization of a von Neumann machine
- Instruction execution in the von Neumann machine
- Instruction representation at both the machine level and in the context of a symbolic assembler
- Simple assembly language programming
- Implementation of functional high level language programming constructs in machine level language
- Main types of memory technology
- Principles of memory management
- Network standards and standardization bodies
- Responsibilities of the first four layers of the ISO reference model
- Fundamentals of cryptography
- Sound, and audio, image and graphics, animation and video
- Multimedia standards
- Different devices used with multimedia
  - Scanners
  - Digital cameras
  - Touch Screens
  - MIDI keyboards and synthesizers
  - Storage devices
- Interaction styles and interaction techniques
- Good design principles for each of the following:
  - Common widgets
  - Sequenced stream presentations
  - Simple error trap dialog
  - On-line help
- Event-driven versus more traditional procedural control for the user interface
- HCI issues in individual versus group interaction
- Social concerns of collaborative software
- HCI issues embodying human intention
- Synchronous versus asynchronous communication
- History of Artificial Intelligence
  - Turing test
- Software life-cycle

Knowledge Area	*Units Covered*	*Suggested Hours*
PF	Fundamental programming constructs	2
	Fundamental data structures	4
	Recursion	3
AL	Basic algorithm analysis	6
	Fundamental computing algorithms	6
AR	Digital logic and digital systems	2
	Assembly level machine organization	2
	Functional organization	1
NC	Communication and networking	2
	Network security	1
	Multimedia Networking Technologies	2
HC	Building a simple graphical user interface	2
	Graphical user-interface design	2
GV	Fundamental techniques of graphing	1
	Graphic systems	1
SE	Software processes	1
Other	Topics of local interest	1
	TOTAL:	**40**

Notes:

This course represents the third and final semester of a breadth-first introductory track that covers fundamental programming concepts in three semesters. Although covering programming fundamentals in two semesters has long been standard in computer science education, more and more programming topics can legitimately be identified as fundamental, making it more difficult to provide a complete introduction to this material in a single year. In terms of the curriculum, students should be able to move on to more advanced computer science courses after taking course sequence CS101B-102B-103B.

This third computer science course should be taken in parallel with the second discrete mathematics course, CS 106, to ensure that the appropriate mathematical foundations of computer science are covered along with the applications of those topics.

The intent of this introductory course will be to contain an integrated laboratory component. The lab component provides the important hands-on programming experience that is vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

3.5 Mathematics Courses

Students should complete a two-course discrete math sequence, as outlined below. The learning objectives associated with discrete mathematics support this degree program. A mathematics department or a computing department (or jointly) should deliver the courses with that intent.

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*
DS	Functions, relations, and sets	9
	Basic logic	5
	Proof techniques	4
	Basics of counting	9
	Interpreting descriptive statistics	9
AR	Digital logic and digital systems	3
SP	Methods and tools of analysis	1
	TOTAL:	**40**

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*
DS	Basic logic	7
	Proof techniques	8
	Graphs and trees	4
	Discrete probability	6
AL	Basic algorithmic analysis	2
	Basic computability	3
	The complexity classes P and NP	2
Other	Matrices	3
	Topics of local interest	5
	TOTAL:	**40**

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.

3.6 Electives

Four career-enhancement electives are detailed in this section. Students are best served by completing one of the three-course sequences CS101, CS102, and CS103, along with career-enhancement electives to provide greater depth in an application area of computing. The local institution should decide which courses are most appropriate, depending on the needs of local industries, requirements of transfer institutions, expertise of the faculty, and availability of hardware and software. Students will develop a reasonable level of understanding in the various subject areas that define the discipline, as well as develop an appreciation for the interrelationships among them. With one or more of these additional electives, students should be better prepared to either successfully transfer to a bachelor's degree program or directly enter the workforce in computing.

Elective: Introduction to Web-Centric Computing

This course introduces students to the World Wide Web environment and Web scripting and management. Topics include: security, privacy, HTML scripting, multimedia, network operating systems, networking protocols, databases, compression and decompression. Students will program using a scripting language, such as JavaScript.

Prerequisite: Mathematical preparation sufficient to qualify for precalculus at the college level.

Syllabus:

- Communication and networking: Overview of network standards and protocols; circuit switching vs. packet switching
- Introduction to the World-Wide Web: Web technologies; the HTML protocol; the format of a web page; support tools for web site creation
- Multimedia data technologies: Sound and audio, image and graphics, animation and video; input and output devices; tools to support multimedia development
- Interactivity on the web: Scripting languages; the role of applets.
- Network security management: Overview of the issues of network management; use of passwords and access control mechanisms; domain names and name services; issues for Internet service providers; security issues and firewalls.
- Fundamentals of cryptography; public-key/private concepts; authentication protocols; digital signatures; digital certificates.
- Compression and decompression: Analog and digital representations; overview of encoding and decoding algorithms; lossless and lossy compression.
- Privacy and civil liberties: Ethical and legal basis for privacy protection; freedom of expression in cyberspace; international and intercultural implications.
- Networking operating systems: basic principles, structuring methods; abstractions, processes, and resources; design of application programming interfaces (API).
- Databases: history and motivation for database systems; components of database systems; DBMS functions; database architecture and data independence; overview of database languages; SQL; query optimization; 4th-generation

environments; embedding non-procedural queries in a procedural language; introduction to Object Query Language.

- Hypertext and hypermedia: Explain basic hypertext and hypermedia concepts; compare and contrast hypermedia delivery based on protocols and systems used; design and implement web-enabled information retrieval applications using appropriate authoring tools.

Knowledge Area	***Units Covered***	***Suggested Hours***
NC	Introduction to net-centric computing	2
	Communication and networking	2
	Network security	3
	The web as an example of client-server computing	3
	Building web applications	3
	Network [security] management	2
	Compression and decompression	3
	Multimedia data technologies	3
HC	Graphical user-interface design	2
	HCI aspects of multimedia systems	1
SE	Software tools and environments	1
OS	Overview of operating systems	1
	Security and protection	2
IM	Database systems	3
	Database query languages	3
	Hypertext and hypermedia	3
Other	Topics of local interests	3
	TOTAL:	**40**

Notes:

This elective course could contain an integrated laboratory component using a multimedia HTML editor and server such as Macromedia Dreamweaver and ColdFusion along with a database such as Microsoft Access. The laboratory component provides an important hands-on experience vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

Topics and experiments that could be included in a laboratory component are:

- Creation of a database driven multimedia Web site.
- Creation of scripts to query the database.
- Use of authentication protocols and digital signatures to provide network and data security.
- Comparison of various compression and decompression algorithms for different types of data, such as text, graphics, sound, and video.
- Creation of a simple usability evaluation for human-computer interaction issues.

Elective: Net-Centric Operating Systems

This course introduces the fundamentals of operating systems design and implementation. Topics include an overview of the components of an operating system, mutual exclusion and synchronization, implementation of processes, scheduling algorithms, memory management, and file systems. It introduces the structured, implementation, and the theoretical underpinnings of computer networking and the applications enabled by that technology.

Prerequisite: CS 102

Syllabus:

- Concurrency: The idea of concurrent execution; states and state diagrams; implementation structures (ready lists, process control blocks, and so forth); dispatching and context switching; interrupt handling in a concurrent environment.
- Mutual exclusion: Definition of the "mutual exclusion" problem; deadlock detection and prevention; solution strategies; models and mechanisms (semaphores, monitors, condition variables, rendezvous); producer-consumer problems; synchronization; multiprocessor issues.
- Scheduling: Preemptive and non-preemptive scheduling; scheduling policies; processes and threads; real-time issues.
- Memory management: Review of physical memory and memory management hardware; overlays, swapping, and partitions; paging and segmentation; page placement and replacement policies; working sets and thrashing; caching.
- File systems: Fundamental concepts (data, metadata, operations, organization, buffering, sequential vs. nonsequential files); content and structure of directories; file system techniques (partitioning, mounting and unmounting, virtual file systems); memory-mapped files; special-purpose file systems; naming, searching, and access; backup strategies.
- Security and protection: Overview of system security; policy/mechanism separation; security methods and devices; protection, access, and authentication; models of protection; memory protection; encryption; recovery management.
- Communication and networking: Network standards and standardization bodies; the ISO 7-layer reference model in general and its instantiation in TCP/IP; circuit switching and packet switching; streams and datagrams; physical layer networking concepts; data link layer concepts; Internetworking and routing; transport layer services.
- Building Web applications: Protocols at the application layer; principles of web engineering web sites; remote procedure calls; lightweight distributed objects; the role of middleware; support tools; security issues in distributed object systems; enterprise-wide web-based applications.
- Network Management: Review of the issues of network management; issues for Internet Service Providers (ISPs); security issues and firewalls; quality of service issues.

- Compression and Decompression: Review of basic data compression; audio compression and decompression; image compression and decompression; video compression and decompression; performance issues.
- Multimedia Data Technologies: Review of multimedia technologies; multimedia standards; capacity planning and performance issues; input and output devices; MIDI keyboards, synthesizers; storage standards; multimedia servers and file systems; tools to support multimedia development.
- Architecture for networks and distributed systems: Introduction to LANs and WANs; layered protocol design, ISO/OSI, IEEE 802; impact of architectural issues on distributed algorithms; network computing; distributed multimedia.

Knowledge Area	***Units Covered***	***Suggested Hours***
OS	Concurrency	3
	Scheduling and dispatch	3
	Memory management	3
	Device management	1
	File systems	3
	Scripting	3
NC	Introduction to net-centric	1
	Communication and networking	3
	Network security	1
	The Web as an example of client-server computing	2
	Building Web applications	6
	Network Management	2
	Compression and Decompression	1
	Multimedia Data Technologies	2
AR	Architecture for networks and distributed systems	3
Other	Topics of local interest	3
	TOTAL:	**40**

Notes:

This elective course could contain an integrated laboratory component using a recent version of Linux, Unix, or Microsoft Windows operating systems and the Java programming language. Emphasis should be placed on the built-in networking and security capabilities of modern operating systems. Simple and secure network socket scripting can be accomplished with built-in methods from the Java networking class.

The laboratory component provides an important hands-on experience vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

Topics and experiments that could be included in a laboratory component are:

- Demonstration of a small network.
- Implementation/scripting of network socket and port security.
- Varying middleware techniques of distributed operating systems.
- Comparison of various compression and decompression algorithms for different data types, such as text, graphics, sound, and video.
- The importance and configuration of virtual memory.

Elective: Computer Organization and Architecture

Topics include: Basic concepts of computer organization and architecture; machine language principles; computer memory organization; I/O fundamentals; and basic operating system concepts and functionality including task scheduling and memory management.

Prerequisite: CS 103

Syllabus:

- Basic organization and functionality of von Neumann machines.
- Representation and execution of machine instructions, implementation of machine language constructs from higher level programming statements.
- Assembly language and machine language coding.
- Computer memory organization and management.
- Data communication and transfer, interrupts, I/O fundamentals.
- Operation of the control unit, instruction management.
- Multiprocessing and alternative architectures.
- Definition and functionality of modern OS.
- Operating principles, scheduling of tasks in operating systems.
- Basic memory management by an operating system.
- Introduction to language translation: comparison of interpreters and compilers; language phases; machine dependent and machine independent aspects of translation; language translation as a software engineering activity.
- Language based constructs for parallelism: communication primitives for tasking models with explicit communication; communication primitives for tasking models for shared memory; programming primitives for data parallel models; comparison of language features for parallel and distributed processing; optimistic concurrency control vs. locking and transactions; asynchronous remote procedure calls.
- Architecture for networks and distributed systems: Introduction to LANs and WANs; layered protocol design, ISO/OSI, IEEE 802; impact of architectural issues on distributed algorithms; network computing; distributed multimedia.

Knowledge Area	*Units Covered*	*Suggested Hours*
AR	Digital logic and digital systems	2
	Machine level representation of data	2
	Assembly level machine organization	9
	Memory system organization and architecture	5
	Interfacing and communication	3
	Functional organization	2
	Multiprocessing and alternative architectures	3
	Architecture for networks and distributed systems	4
OS	Overview of operating systems	1
	Operating systems principles	1
	Scheduling and dispatch	2
	Memory management	1
PL	Introduction to languages translation	2
Other	Topics of local interest	3
	TOTAL:	**40**

Notes:

This elective course could contain an integrated laboratory component that provides an important hands-on experience vital for beginning computer science students. The forty (40) total hours listed above are considered "lecture" hours and typically two (2) hours of lab equates to one (1) hour of lecture.

Topics and experiments that could be included in a laboratory component are:

- Design, implement and test combinational circuits.
- Use device management concepts.
- Use file systems and their naming conventions.
- Apply process scheduling and dispatching.
- Analyze physical and virtual memory organization.
- Design system security and protection.
- Work with distributed and real-time systems.
- Analyze computer system organization.
- Experiment with assembly language programming.
- Analyze selected components of a compiler or interpreter.
- Work with low-level I/O operations.
- Work with basic principles of operating systems.
- Implement parallelism in a programming application.

Elective: Hardware Fundamentals

The topics are centered around a hardware-oriented introduction to the computer, beginning with logic and its implementation in logic gates, and moving up the conceptual hierarchy through assembly language in preparation for teaching a higher-level language. This course contains hands on lab exercises to provide the student with an introduction to the hardware components within a microcomputer, including a lab component to build a microcomputer.

Prerequisite: Mathematical preparation sufficient to qualify for precalculus at the college level.

Syllabus:

- Introduction: history of computing from a hardware perspective, overview of a computer system, computers as universal computational devices
- Introduction to propositional logic, logical connectives and truth tables
- Fundamental building blocks such as logic gates, flip-flops, counters, registers, and PLA's
    - Lab component: Building and demonstrating fundamental logic circuits.
- Normal forms (conjunctive and disjunctive), validity, logic expressions, minimization, sum of product forms, notions of implication, converse, contrapositive, negation, and contradiction
- Physical considerations such as gate delays, fan-in and fan-out
    - Lab component: Measurement of logic circuit signals, test equipment
- Numeric data representation: Bits, bytes, and words, number bases, fixed and floating point number systems, signed and two's complement representations
- Representation of nonnumeric data such as character codes and graphical data
- The von Neumann machine, control units, instruction fetch, decode, and execution
    - Lab component: Using a simple microprocessor breadboard system with monitor control.
- Instruction sets and types, instruction formats, addressing modes, segmentation of memory
- Write simple assembly language program segments
    - Lab component: Implementing a simple assembly language program on a microprocessor
- Subroutine calls and return mechanisms
- Role, functionality and history of the operating system
    - Lab component: Implementing a more complex assembly language program with operating system subroutine calls on a microcomputer.
- Storage systems, memory hierarchy, main memory organization and operation, cache memories and virtual memories
- I/O and Interrupts
    - Lab component: Interfacing a support chip such as a UART
- Lab Capstone Project: Building an operational microcomputer.

Knowledge Area	***Units Covered***	***Suggested Hours***
PF	Fundamental Data Structures	1
DS	Basic Logic	5
	Proof Techniques	2
AR	Digital Logic and Digital Systems	6
	Machine Level Representation of Data	3
	Assembly Level Machine Organization	9
	Memory System Organization and Architecture	3
OS	Overview of Operating Systems	1
SP	History of Computing	1
Other	Topics of local interest	2
Lab	Lab hours	7
	TOTAL:	**40**

Notes:

This course is an introductory hardware oriented course, which is not designed as a transferable course in either Assembly Language or Architecture.

Acknowledgements

Task Force Members

The ACM Two-Year College Education Committee gratefully acknowledges the outstanding work done by the following Task Force members in the development of this report.

Anne Applin, Pearl River Community College (MS)
Scott Badman, Parkland Community College (IL)
Julia Benson, Georgia Perimeter College (GA)
Elizabeth "Beth" Hawthorne, Union County College (NJ)
Josephine Freedman, Suffolk County Community College (NY)
Wayne Horn, Pensacola Junior College (FL)
Carol Janik, Tompkins Cortland Community College (NY)
Keith Jolly, Las Positas College (CA)
Timothy Klingler, Delta College (MI)
Dan Myers, Sun Microsystems, Inc. (CO)
Holly Patterson-McNeill, Texas A&M - Corpus Christi (TX)
T.S. Pennington, Longview Community College (MO)
Judy Porter, Cape Fear Community College (NC)
Alfred Shin, Humber College (Canada)
Sylvia Sorkin, Community College of Baltimore County (MD)
Ed Wilkens, Salem State College (MA)
Anita Wright, Camden County College (NJ)

Special Thanks

The ACM Two-Year College Education Committee would like to thank Carl K. Chang (Vice-President, IEEE-CS Education Activities Board) and Peter J. Denning (Chair, ACM Education Board) for their support of this undertaking. Special appreciation is also expressed to the IEEE-CS/ACM CS2001 Steering Committee - especially Eric Roberts and Russell Shackelford - for their generous cooperation, collaboration and support in the development of this document. Appreciation is also expressed to John Impagliazzo for his valuable assistance in editing the final document.

Bibliography and References

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

ACM Two-Year College Education Committee, *Guidelines for Associate Degree Programs to Support Computing in a Networked Environment*, 2000.

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).

Davis, Gorgone, Couger, Feinstein, and Longnecker, *IS'97 model curriculum and guidelines for undergraduate degree programs in information systems, Association of Information Technology Professionals*, (1997).

Epp, Susanna S., *Discrete Mathematics with Applications*, 2nd edition, Brooks/Cole Publishing (1995)

Freeman and Aspray, *The Supply of Information Technology Workers in the United States*, Computing Research Association, NSF grant EIA 9812240 (1999).

National Science Foundation Advisory Committee, *Shaping the Future: New Expectations for Undergraduate Education in Science, Mathematics, Engineering and Technology*, NSF (1999).

Appendix A. Taxonomy of Learning Processes

Table A.1 is an adaptation of Bloom's Taxonomy. It shows the taxonomy levels in ascending order with a definition for each level. The table also includes verbs that may be useful in the design of course activities.

Table A.1 - Bloom's Taxonomy, Modified

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