

COMPUTER SCIENCE BODY OF KNOWLEDGE

Table of Content

Discrete Structures (DS)	6
DS1. Functions, relations, and sets [core]	6
DS1. Functions, relations, and sets [core]	6
DS2. Basic logic [core]	7
DS3. Proof techniques [core]	7
DS4. Basics of counting [core]	7
DS5. Graphs and trees [core]	8
DS6. Discrete probability [core]	8
Programming Fundamentals (PF)	10
PF1. Fundamental programming constructs [core]	10
PF2. Algorithms and problem-solving [core]	11
PF3. Fundamental data structures [core]	11
PF4. Recursion [core]	11
PF5. Event-driven programming [core]	12
Algorithms and Complexity (AL)	13
AL1. Basic algorithmic analysis [core]	13
AL2. Algorithmic strategies [core]	14
AL3. Fundamental computing algorithms [core]	14
AL4. Distributed algorithms [core]	15
AL5. Basic computability [core]	15
AL6. The complexity classes P and NP [elective]	15
AL7. Automata theory [elective]	16
AL8. Advanced algorithmic analysis [elective]	16
AL9. Cryptographic algorithms [elective]	17
AL10. Geometric algorithms [elective]	17
AL11. Parallel algorithms [elective]	17
Architecture and Organization (AR)	18
AR1. Digital logic and digital systems [core]	18
AR2. Machine level representation of data [core]	19
AR3. Assembly level machine organization [core]	19
AR4. Memory system organization and architecture [core]	19
AR5. Interfacing and communication [core]	20
AR6. Functional organization [core]	20
AR7. Multiprocessing and alternative architectures [core]	21
AR8. Performance enhancements [elective]	21
AR9. Architecture for networks and distributed systems [elective]	21
Operating Systems (OS)	23
OS1. Overview of operating systems [core]	23
OS2. Operating system principles [core]	24
OS3. Concurrency [core]	24
OS4. Scheduling and dispatch [core]	25
OS5. Memory management [core]	25
OS6. Device management [elective]	26

OS7. Security and protection [elective]	26
OS8. File systems [elective].....	27
OS9. Real-time and embedded systems [elective]	27
OS10. Fault tolerance [elective].....	27
OS11. System performance evaluation [elective]	28
OS12. Scripting [elective].....	28
Net-Centric Computing (NC).....	29
NC1. Introduction to net-centric computing [core].....	29
NC2. Communication and networking [core]	29
NC3. Network security [core]	30
NC4. The web as an example of client-server computing [core].....	30
NC5. Building web applications [elective]	31
NC6. Network management [elective].....	31
NC7. Compression and decompression [elective]	32
NC8. Multimedia data technologies [elective].....	32
NC9. Wireless and mobile computing [elective]	33
Programming Languages (PL)	34
PL1. Overview of programming languages [core].....	34
PL2. Virtual machines [core]	35
PL3. Introduction to language translation [core]	35
PL4. Declarations and types [core]	35
PL5. Abstraction mechanisms [core]	36
PL6. Object-oriented programming [core].....	36
PL7. Functional programming [elective]	36
PL8. Language translation systems [elective].....	37
PL9. Type systems [elective]	37
PL10. Programming language semantics [elective].....	38
PL11. Programming language design [elective]	38
Human-Computer Interaction (HC)	39
HC1. Foundations of human-computer interaction [core]	39
HC2. Building a simple graphical user interface [core].....	39
HC3. Human-centered software evaluation [elective]	40
HC4. Human-centered software development [elective]	40
HC5. Graphical user-interface design [elective]	40
HC6. Graphical user-interface programming [elective].....	41
HC7. HCI aspects of multimedia systems [elective].....	41
HC8. HCI aspects of collaboration and communication [elective].....	42
Graphics and Visual Computing (GV).....	43
GV1. Fundamental techniques in graphics [core]	43
GV2. Graphic systems [core]	44
GV3. Graphic communication [elective]	44
GV4. Geometric modeling [elective]	45
GV5. Basic rendering [elective].....	45
GV6. Advanced rendering [elective]	45
GV7. Advanced techniques [elective].....	46
GV8. Computer animation [elective]	46
GV9. Visualization [elective].....	47
GV10. Virtual reality [elective]	47

GV11. Computer vision [elective]	47
Intelligent Systems (IS)	49
IS1. Fundamental issues in intelligent systems [core]	49
IS2. Search and constraint satisfaction [core]	50
IS3. Knowledge representation and reasoning [core]	50
IS4. Advanced search [elective]	50
IS5. Advanced knowledge representation and reasoning [elective]	51
IS6. Agents [elective]	51
IS7. Natural language processing [elective]	52
IS8. Machine learning and neural networks [elective]	53
IS9. AI planning systems [elective]	53
IS10. Robotics [elective]	54
Information Management (IM).....	55
IM1. Information models and systems [core]	55
IM2. Database systems [core]	56
IM3. Data modeling [core]	56
IM4. Relational databases [elective]	56
IM5. Database query languages [elective]	57
IM6. Relational database design [elective]	57
IM7. Transaction processing [elective]	58
IM8. Distributed databases [elective]	58
IM9. Physical database design [elective]	58
IM10. Data mining [elective]	59
IM11. Information storage and retrieval [elective]	59
IM12. Hypertext and hypermedia [elective]	60
IM13. Multimedia information and systems [elective]	60
IM14. Digital libraries [elective]	61
Social and Professional Issues (SP).....	62
SP1. History of computing [core]	63
SP2. Social context of computing [core]	63
SP3. Methods and tools of analysis [core]	64
SP4. Professional and ethical responsibilities [core]	64
SP5. Risks and liabilities of computer-based systems [core]	65
SP6. Intellectual property [core]	65
SP7. Privacy and civil liberties [core]	65
SP8. Computer crime [elective]	66
SP9. Economic issues in computing [elective]	66
SP10. Philosophical frameworks [elective]	66
Software Engineering (SE).....	67
SE1. Software design [core]	67
SE2. Using APIs [core]	68
SE3. Software tools and environments [core]	68
SE4. Software processes [core]	68
SE5. Software requirements and specifications [core]	69
SE6. Software validation [core]	69
SE7. Software evolution [core]	70
SE8. Software project management [core]	70
SE9. Component-based computing [elective]	71

SE10. Formal methods [elective]	71
SE11. Software reliability [elective]	72
SE12. Specialized systems development [elective]	72
Computational Science and Numerical Methods (CN)	72
CN1. Numerical analysis [elective]	73
CN2. Operations research [elective]	74
CN3. Modeling and simulation [elective]	74
CN4. High-performance computing [elective].....	75

Figure A-1. Computer science body of knowledge with core topics underlined

<p>DS. Discrete Structures (43 core hours) <u>DS1. Functions, relations, and sets</u> (6) DS2. Basic logic (10) <u>DS3. Proof techniques</u> (12) DS4. Basics of counting (5) DS5. Graphs and trees (4) <u>DS6. Discrete probability</u> (6)</p> <p>PF. Programming Fundamentals (38 core hours) <u>PF1. Fundamental programming constructs</u> (9) PF2. Algorithms and problem-solving (6) <u>PF3. Fundamental data structures</u> (14) <u>PF4. Recursion</u> (5) PF5. Event-driven programming (4)</p> <p>AL. Algorithms and Complexity (31 core hours) <u>AL1. Basic algorithmic analysis</u> (4) <u>AL2. Algorithmic strategies</u> (6) <u>AL3. Fundamental computing algorithms</u> (12) <u>AL4. Distributed algorithms</u> (3) <u>AL5. Basic computability</u> (6) AL6. The complexity classes P and NP AL7. Automata theory AL8. Advanced algorithmic analysis AL9. Cryptographic algorithms AL10. Geometric algorithms AL11. Parallel algorithms</p> <p>AR. Architecture and Organization (36 core hours) <u>AR1. Digital logic and digital systems</u> (6) <u>AR2. Machine level representation of data</u> (3) <u>AR3. Assembly level machine organization</u> (9) <u>AR4. Memory system organization and architecture</u> (5) <u>AR5. Interfacing and communication</u> (3) <u>AR6. Functional organization</u> (7) <u>AR7. Multiprocessing and alternative architectures</u> (3) AR8. Performance enhancements AR9. Architecture for networks and distributed systems</p> <p>OS. Operating Systems (18 core hours) <u>OS1. Overview of operating systems</u> (2) <u>OS2. Operating system principles</u> (2) <u>OS3. Concurrency</u> (6) <u>OS4. Scheduling and dispatch</u> (3) <u>OS5. Memory management</u> (5) 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</p> <p>NC. Net-Centric Computing (15 core hours) <u>NC1. Introduction to net-centric computing</u> (2) <u>NC2. Communication and networking</u> (7) <u>NC3. Network security</u> (3) <u>NC4. The web as an example of client-server computing</u> (3) NC5. Building web applications NC6. Network management NC7. Compression and decompression NC8. Multimedia data technologies NC9. Wireless and mobile computing</p> <p>PL. Programming Languages (21 core hours) <u>PL1. Overview of programming languages</u> (2) <u>PL2. Virtual machines</u> (1) <u>PL3. Introduction to language translation</u> (2) <u>PL4. Declarations and types</u> (3) <u>PL5. Abstraction mechanisms</u> (3) <u>PL6. Object-oriented programming</u> (10) PL7. Functional programming PL8. Language translation systems PL9. Type systems PL10. Programming language semantics PL11. Programming language design</p> <p><i>Note: The numbers in parentheses represent the <u>minimum</u> number of hours required to cover this material in a lecture format. It is always appropriate to include more.</i></p>	<p>HC. Human-Computer Interaction (8 core hours) <u>HC1. Foundations of human-computer interaction</u> (6) <u>HC2. Building a simple graphical user interface</u> (2) 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</p> <p>GV. Graphics and Visual Computing (3 core hours) <u>GV1. Fundamental techniques in graphics</u> (2) <u>GV2. Graphic systems</u> (1) 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</p> <p>IS. Intelligent Systems (10 core hours) <u>IS1. Fundamental issues in intelligent systems</u> (1) <u>IS2. Search and constraint satisfaction</u> (5) <u>IS3. Knowledge representation and reasoning</u> (4) 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</p> <p>IM. Information Management (10 core hours) <u>IM1. Information models and systems</u> (3) <u>IM2. Database systems</u> (3) <u>IM3. Data modeling</u> (4) 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</p> <p>SP. Social and Professional Issues (16 core hours) <u>SP1. History of computing</u> (1) <u>SP2. Social context of computing</u> (3) <u>SP3. Methods and tools of analysis</u> (2) <u>SP4. Professional and ethical responsibilities</u> (3) <u>SP5. Risks and liabilities of computer-based systems</u> (2) <u>SP6. Intellectual property</u> (3) <u>SP7. Privacy and civil liberties</u> (2) SP8. Computer crime SP9. Economic issues in computing SP10. Philosophical frameworks</p> <p>SE. Software Engineering (31 core hours) <u>SE1. Software design</u> (8) <u>SE2. Using APIs</u> (5) <u>SE3. Software tools and environments</u> (3) <u>SE4. Software processes</u> (2) <u>SE5. Software requirements and specifications</u> (4) <u>SE6. Software validation</u> (3) <u>SE7. Software evolution</u> (3) <u>SE8. Software project management</u> (3) SE9. Component-based computing SE10. Formal methods SE11. Software reliability SE12. Specialized systems development</p> <p>CN. Computational Science (no core hours) CN1. Numerical analysis CN2. Operations research CN3. Modeling and simulation CN4. High-performance computing</p>
---	--

Discrete Structures (DS)

DS1. Functions, relations, and sets [core]

DS2. Basic logic [core]

DS3. Proof techniques [core]

DS4. Basics of counting [core]

DS5. Graphs and trees [core]

DS6. Discrete probability [core]

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. Discrete structures includes important material from such areas as set theory, logic, graph theory, and combinatorics.

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. Graph theory concepts are used in networks, operating systems, and compilers. Set theory concepts are used in software engineering and in databases. As the field of computer science matures, more and more 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. We have 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 great 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.

DS1. Functions, relations, and sets [core]

Minimum core coverage time: 6 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.

DS2. Basic logic [core]

Minimum core coverage time: 10 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. Describe 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. Describe the importance and limitations of predicate logic.

DS3. Proof techniques [core]

Minimum core 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 described in this unit.
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.

DS4. Basics of counting [core]

Minimum core coverage time: 5 hours

Topics:

Counting arguments

- Sum and product rule
- Inclusion-exclusion principle
- Arithmetic and geometric progressions
- Fibonacci numbers
- The pigeonhole principle
- Permutations and combinations
- Basic definitions
- Pascal's identity
- The binomial theorem

Solving recurrence relations

- Common examples
- The Master theorem

Learning objectives:

1. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
2. State the definition of the Master theorem.
3. Solve a variety of basic recurrence equations.
4. Analyze a problem to create relevant recurrence equations or to identify important counting questions.

DS5. Graphs and trees [core]

Minimum core 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. Demonstrate 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.

DS6. Discrete probability [core]

Minimum core coverage time: 6 hours

Topics:

- Finite probability space, probability measure, events
- Conditional probability, independence, Bayes' theorem
- 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.

Programming Fundamentals (PF)

PF1. Fundamental programming constructs [core]

PF2. Algorithms and problem-solving [core]

PF3. Fundamental data structures [core]

PF4. Recursion [core]

PF5. Event-driven programming [core]

Fluency in a programming language is prerequisite to the study of most of computer science. In the CC1991 report, knowledge of a programming language—while identified as essential—was given little emphasis in the curriculum. The “Introduction to a Programming Language” area in CC1991 represents only 12 hours of class time and is identified as optional, under the optimistic assumption that “increasing numbers of students . . . gain such experience in secondary school.” We believe that undergraduate computer science programs must teach students how to use at least one programming language well; furthermore, we recommend that computer science programs should teach students to become competent in languages that make use of at least two programming paradigms. Accomplishing this goal requires considerably more than 12 hours. 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 equally well have been assigned to either Programming Fundamentals or the more advanced area.

PF1. Fundamental programming constructs [core]

Minimum core coverage time: 9 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 covered by this unit.
2. Modify and expand short programs that use standard conditional and iterative control structures and functions.
3. 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.
4. Choose appropriate conditional and iteration constructs for a given programming task.
5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
6. Describe the mechanics of parameter passing.

PF2. Algorithms and problem-solving [core]

Minimum core coverage time: 6 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 the importance of algorithms in the problem-solving process.
2. Identify the necessary properties of good algorithms.
3. Create algorithms for solving simple problems.
4. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems.
5. Describe strategies that are useful in debugging.

PF3. Fundamental data structures [core]

Minimum core coverage time: 14 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. Discuss the representation and use of primitive data types and built-in data structures.
2. Describe how the data structures in the topic list are allocated and used in memory.
3. Describe common applications for each data structure in the topic list.
4. Implement the user-defined data structures in a high-level language.
5. Compare alternative implementations of data structures with respect to performance.
6. Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.
7. Compare and contrast the costs and benefits of dynamic and static data structure implementations.
8. Choose the appropriate data structure for modeling a given problem.

PF4. Recursion [core]

Minimum core coverage time: 5 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 the concept of recursion and give examples of its use.
2. Identify the base case and the general case of a recursively defined problem.
3. Compare iterative and recursive solutions for elementary problems such as factorial.
4. Describe the divide-and-conquer approach.
5. Implement, test, and debug simple recursive functions and procedures.
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.

PF5. Event-driven programming [core]

Minimum core coverage time: 4 hours

Topics:

- Event-handling methods
- Event propagation
- Exception handling

Learning objectives:

1. Explain the difference between event-driven programming and command-line programming.
2. Design, code, test, and debug simple event-driven programs that respond to user events.
3. Develop code that responds to exception conditions raised during execution.

Algorithms and Complexity (AL)

- AL1. Basic algorithmic analysis [core]
- AL2. Algorithmic strategies [core]
- AL3. Fundamental computing algorithms [core]
- AL4. Distributed algorithms [core]
- AL5. Basic computability [core]
- AL6. The complexity classes P and NP [elective]
- AL7. Automata theory [elective]
- AL8. Advanced algorithmic analysis [elective]
- AL9. Cryptographic algorithms [elective]
- AL10. Geometric algorithms [elective]
- AL11. Parallel algorithms [elective]

Algorithms are fundamental to computer science and software engineering. The realworld 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.

AL1. Basic algorithmic analysis [core]

Minimum core coverage time: 4 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.

AL2. Algorithmic strategies [core]

Minimum core coverage time: 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. For each of several kinds of algorithm (brute force, greedy, divide-and-conquer, backtracking, branch-and-bound, and heuristic), identify an example of everyday human behavior that exemplifies the basic concept.
3. Implement a greedy algorithm to solve an appropriate problem.
4. Implement a divide-and-conquer algorithm to solve an appropriate problem.
5. Use backtracking to solve a problem such as navigating a maze.
6. Describe various heuristic problem-solving methods.
7. Use pattern matching to analyze substrings.
8. Use numerical approximation to solve mathematical problems, such as finding the roots of a polynomial.

AL3. Fundamental computing algorithms [core]

Minimum core coverage time: 12 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. Implement the most common quadratic and $O(N \log N)$ sorting algorithms.
2. Design and implement an appropriate hashing function for an application.
3. Design and implement a collision-resolution algorithm for a hash table.

4. Discuss the computational efficiency of the principal algorithms for sorting, searching, and hashing.
5. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of applicationspecific patterns in the input data.
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 capabilities: 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.

AL4. Distributed algorithms [core]

Minimum core coverage time: 3 hours

Topics:

- Consensus and election
- Termination detection
- Fault tolerance
- Stabilization

Learning objectives:

1. Explain the distributed paradigm.
2. Explain one simple distributed algorithm.
3. Determine when to use consensus or election algorithms.
4. Distinguish between logical and physical clocks.
5. Describe the relative ordering of events in a distributed algorithm.

AL5. Basic computability [core]

Minimum core coverage time: 6 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. Explain how some problems have no algorithmic solution.
5. Provide examples that illustrate the concept of uncomputability.

AL6. The complexity classes P and NP [elective]

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.

AL7. Automata theory [elective]

Topics:

- Deterministic finite automata (DFAs)
- Nondeterministic finite automata (NFAs)
- Equivalence of DFAs and NFAs
- Regular expressions
- The pumping lemma for regular expressions
- Push-down automata (PDAs)
- Relationship of PDAs and context-free grammars
- Properties of context-free grammars
- Turing machines
- Nondeterministic Turing machines
- Sets and languages
- Chomsky hierarchy
- The Church-Turing thesis

Learning objectives:

1. Determine a language's location in the Chomsky hierarchy (regular sets, context-free, context-sensitive, and recursively enumerable languages).
2. Prove that a language is in a specified class and that it is not in the next lower class.
3. Convert among equivalently powerful notations for a language, including among DFAs, NFAs, and regular expressions, and between PDAs and CFGs.
4. Explain at least one algorithm for both top-down and bottom-up parsing.
5. Explain the Church-Turing thesis and its significance.

AL8. Advanced algorithmic analysis [elective]

Topics:

- Amortized analysis
- Online and offline algorithms
- Randomized algorithms
- Dynamic programming
- Combinatorial optimization

)

Learning objectives:

1. Use the potential method to provide an amortized analysis of previously unseen data structure, given the potential function.
2. Explain why competitive analysis is an appropriate measure for online algorithms.
3. Explain the use of randomization in the design of an algorithm for a problem where a deterministic algorithm is unknown or much more difficult.

4. Design and implement a dynamic programming solution to a problem.

AL9. Cryptographic algorithms [elective]

Topics:

- Historical overview of cryptography
- Private-key cryptography and the key-exchange problem
- Public-key cryptography
- Digital signatures
- Security protocols
- Applications (zero-knowledge proofs, authentication, and so on)

Learning objectives:

1. Describe efficient basic number-theoretic algorithms, including greatest common divisor, multiplicative inverse mod n , and raising to powers mod n .
2. Describe at least one public-key cryptosystem, including a necessary complexitytheoretic assumption for its security.
3. Create simple extensions of cryptographic protocols, using known protocols and cryptographic primitives.

AL10. Geometric algorithms [elective]

Topics:

- Line segments: properties, intersections
- Convex hull finding algorithms

Learning objectives:

1. Describe and give time analysis of at least two algorithms for finding a convex hull.
2. Justify the $\Omega(N \log N)$ lower bound on finding the convex hull.
3. Describe at least one additional efficient computational geometry algorithm, such as finding the closest pair of points, convex layers, or maximal layers.

AL11. Parallel algorithms [elective]

Topics:

- PRAM model
- Exclusive versus concurrent reads and writes
- Pointer jumping
- Brent's theorem and work efficiency

Learning objectives:

1. Describe implementation of linked lists on a PRAM.
2. Use parallel-prefix operation to perform simple computations efficiently in parallel.
3. Explain Brent's theorem and its relevance.

Architecture and Organization (AR)

- AR1. Digital logic and digital systems [core]
- AR2. Machine level representation of data [core]
- AR3. Assembly level machine organization [core]
- AR4. Memory system organization and architecture [core]
- AR5. Interfacing and communication [core]
- AR6. Functional organization [core]
- AR7. Multiprocessing and alternative architectures [core]
- AR8. Performance enhancements [elective]
- AR9. Architecture for networks and distributed systems [elective]

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 be able to understand the tradeoff among various components, such as CPU clock speed vs. memory size.

The learning outcomes specified for these topics correspond primarily to the core and are intended to support programs that elect to require only the minimum 36 hours of computer architecture of their students. For programs that want to teach more than the minimum, the same topics (AR1-AR7) can be treated at a more advanced level by implementing a two-course sequence. For programs that want to cover the elective topics, those topics can be introduced within a two-course sequence and/or be treated in a more comprehensive way in a third course.

AR1. Digital logic and digital systems [core]

Minimum core coverage time: 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 and their role in 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.

AR2. Machine level representation of data [core]

Minimum core coverage time: 3 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 reasons for using different formats to represent numerical data.
2. Explain how negative integers are stored in sign-magnitude and twos-complement representation.
3. Convert numerical data from one format to another.
4. Discuss how fixed-length number representations affect accuracy and precision.
5. Describe the internal representation of nonnumeric data.
6. Describe the internal representation of characters, strings, records, and arrays.

AR3. Assembly level machine organization [core]

Minimum core coverage time: 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 an instruction is executed 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 simple assembly language program segments.
6. Demonstrate how fundamental high-level programming constructs are implemented at the machine-language level.
7. Explain how subroutine calls are handled at the assembly level.
8. Explain the basic concepts of interrupts and I/O operations.

AR4. Memory system organization and architecture [core]

Minimum core coverage time: 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.

AR5. Interfacing and communication [core]

Minimum core coverage time: 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 interrupts are used 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.
4. Compare the common network configurations.
5. Identify interfaces needed for multimedia support.
6. Describe the advantages and limitations of RAID architectures.

AR6. Functional organization [core]

Minimum core coverage time: 7 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.

AR7. Multiprocessing and alternative architectures [core]

Minimum core coverage time: 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.
3. Explain the concept of interconnection networks and characterize different approaches.
4. Discuss the special concerns that multiprocessing systems present with respect to memory management and describe how these are addressed.

AR8. Performance enhancements [elective]

Topics:

- Superscalar architecture
- Branch prediction
- Prefetching
- Speculative execution
- Multithreading
- Scalability

Learning objectives:

1. Describe superscalar architectures and their advantages.
2. Explain the concept of branch prediction and its utility.
3. Characterize the costs and benefits of prefetching.
4. Explain speculative execution and identify the conditions that justify it.
5. Discuss the performance advantages that multithreading can offer in an architecture along with the factors that make it difficult to derive maximum benefits from this approach.
6. Describe the relevance of scalability to performance.

AR9. Architecture for networks and distributed systems [elective]

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. Discuss the architectural issues involved in the design of a layered network protocol.
3. Explain how architectures differ in network and distributed systems.
4. Discuss architectural issues related to network computing and distributed multimedia.

Operating Systems (OS)

- OS1. Overview of operating systems [core]
- OS2. Operating system principles [core]
- OS3. Concurrency [core]
- OS4. Scheduling and dispatch [core]
- OS5. Memory management [core]
- OS6. Device management [elective]
- OS7. Security and protection [elective]
- OS8. File systems [elective]
- OS9. Real-time and embedded systems [elective]
- OS10. Fault tolerance [elective]
- OS11. System performance evaluation [elective]
- OS12. Scripting [elective]

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. The topics in this area explain the issues that influence the design of contemporary operating systems. Courses that cover this area will typically include a laboratory component to enable students to experiment with operating systems. 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 prior to a detailed study of internal implementation algorithms and data structures. Therefore these topics address 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.

OS1. Overview of operating systems [core]

Minimum core coverage time: 2 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 have evolved over time from primitive batch systems to sophisticated multiuser 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 issues such as open source software and the increased use of the Internet are influencing operating system design.

OS2. Operating system principles [core]

Minimum core coverage time: 2 hours

Topics:

- Structuring methods (monolithic, layered, modular, micro-kernel models)
- Abstractions, processes, and resources
- Concepts of application program interfaces (APIs)
- Application 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. Defend the need for APIs and middleware.
4. Describe how computing resources are used by application software and managed by system software.
5. Contrast kernel and user mode in an operating system.
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.

OS3. Concurrency [core]

Minimum core coverage time: 6 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 separate tasks.

3. Summarize the range of mechanisms that can be employed at the operating system level 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 various 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 for simple problem domains.
8. Discuss the utility of data structures, such as stacks and queues, in managing concurrency.
9. Explain conditions that lead to deadlock.

OS4. Scheduling and dispatch [core]

Minimum core coverage time: 3 hours

Topics:

- Preemptive and nonpreemptive 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 nonpreemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes.
2. Describe relationships between scheduling algorithms and application domains.
3. Discuss the types of processor scheduling such as short-term, medium-term, longterm, and I/O.
4. Describe the difference between processes and threads.
5. Compare and contrast static and dynamic approaches to real-time scheduling.
6. Discuss the need for preemption and deadline scheduling.
7. Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and other problems unrelated to computing.

OS5. Memory management [core]

Minimum core coverage time: 5 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. Explain memory hierarchy and cost-performance tradeoffs.
2. Explain the concept of virtual memory and how it is realized in hardware and software.
3. Summarize 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, citing the relative merits of each.
6. Describe the reason for and use of cache memory.
7. Compare and contrast paging and segmentation techniques.
8. Discuss the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem.
9. Analyze the various memory portioning techniques including overlays, swapping, and placement and replacement policies.

OS6. Device management [elective]

Topics:

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

Learning objectives:

1. Explain the key difference between serial and parallel devices and identify the conditions in which each is appropriate.
2. Identify the relationship between the physical hardware and the virtual devices maintained by the operating system.
3. Explain buffering and describe strategies for implementing it.
4. Differentiate the mechanisms used in interfacing a range of devices (including handheld devices, networks, multimedia) to a computer and explain the implications of these for the design of an operating system.
5. Describe the advantages and disadvantages of direct memory access and discuss the circumstances in which its use is warranted.
6. Identify the requirements for failure recovery.
7. Implement a simple device driver for a range of possible devices.

OS7. Security and protection [elective]

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.

4. Compare and contrast the strengths and weaknesses of two or more currently popular operating systems with respect to security.
5. Compare and contrast the security strengths and weaknesses of two or more currently popular operating systems with respect to recovery management.

OS8. File systems [elective]

Topics:

- Files: data, metadata, operations, organization, buffering, sequential, nonsequential
- Directories: contents and structure
- File systems: partitioning, mount/unmount, 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, recognizing the strengths and weaknesses of each.
3. Summarize how hardware developments have lead to changes in our priorities for the design and the management of file systems.

OS9. Real-time and embedded systems [elective]

Topics:

Process and task scheduling

- Memory/disk management requirements in a real-time environment
- Failures, risks, and recovery
- Special concerns in real-time systems

Learning objectives:

1. Describe what makes a system a real-time system.
2. Explain the presence of and describe the characteristics of latency in real-time systems.
3. Summarize special concerns that real-time systems present and how these concerns are addressed.

OS10. Fault tolerance [elective]

Topics:

- Fundamental concepts: reliable and available systems
- Spatial and temporal redundancy
- Methods used to implement fault tolerance
- Examples of reliable systems

Learning objectives:

1. Explain the relevance of the terms fault tolerance, reliability, and availability.
2. Outline the range of methods for implementing fault tolerance in an operating system.
3. Explain how an operating system can continue functioning after a fault occurs.

OS11. System performance evaluation [elective]

Topics:

- Why system performance needs to be evaluated
- What is to be evaluated
- Policies for caching, paging, scheduling, memory management, security, and so forth
- Evaluation models: deterministic, analytic, simulation, or implementation-specific
- How to collect evaluation data (profiling and tracing mechanisms)

Learning objectives:

1. Describe the performance metrics used to determine how a system performs.
2. Explain the main evaluation models used to evaluate a system.

OS12. Scripting [elective]

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.

Net-Centric Computing (NC)

- NC1. Introduction to net-centric computing [core]
- NC2. Communication and networking [core]
- NC3. Network security [core]
- NC4. The web as an example of client-server computing [core]
- NC5. Building web applications [elective]
- NC6. Network management [elective]
- NC7. Compression and decompression [elective]
- NC8. Multimedia data technologies [elective]
- NC9. Wireless and mobile computing [elective]

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 be understood by laboratory experimentation.

NC1. Introduction to net-centric computing [core]

Minimum core coverage time: 2 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 the ability to use effectively a range of common networked applications including e-mail, telnet, FTP, newsgroups, and web browsers, online web courses, and instant messaging.
3. Explain the hierarchical, layered structure of a typical network architecture.
4. Describe emerging technologies in the net-centric computing area and assess their current capabilities, limitations, and near-term potential.

NC2. Communication and networking [core]

Minimum core coverage time: 7 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 responsibilities of the first four layers of the ISO reference model.
3. Discuss the differences between circuit 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 how a packet is routed over the Internet.
6. Install a simple network with two clients and a single server using standard hostconfiguration software tools such as DHCP.

NC3. Network security [core]

Minimum core coverage time: 3 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. Describe how public-key cryptography works.
3. Distinguish between the use of private- and public-key algorithms.
4. Summarize common authentication protocols.
5. Generate and distribute a PGP key pair and use the PGP package to send an encrypted e-mail message.
6. Summarize the capabilities and limitations of the means of cryptography that are conveniently available to the general public.

NC4. The web as an example of client-server computing [core]

Minimum core coverage time: 3 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 various 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).

NC5. Building web applications [elective]

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 interactive client-server web applications of medium size can be built 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 and strategies in an enterprise-wide web-based application.

NC6. Network management [elective]

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.

NC7. Compression and decompression [elective]

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, 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.

NC8. Multimedia data technologies [elective]

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. For each of several media or multimedia standards, describe in non-technical language what the standard calls for, and explain how aspects of human perception might be sensitive to the limitations of that standard.
2. 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.
3. 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.
4. Implement a multimedia application of modest size.

NC9. Wireless and mobile computing [elective]

Topics:

Overview of the history, evolution, and compatibility of wireless standards

- The special problems of wireless and mobile computing
- Wireless local area networks and satellite-based networks
- Wireless local loops
- Mobile Internet protocol
- Mobile aware adaption
- Extending the client-server model to accommodate mobility
- Mobile data access: server data dissemination and client cache management
- Software package support for mobile and wireless computing
- The role of middleware and support tools
- Performance issues
- Emerging technologies

Learning objectives:

1. Describe the main characteristics of mobile IP and explain how differs from IP with regard to mobility management and location management as well as performance.
2. Illustrate (with home agents and foreign agents) how e-mail and other traffic is routed using mobile IP.
3. Implement a simple application that relies on mobile and wireless data communications.
4. Describe areas of current and emerging interest in wireless and mobile computing, and assess the current capabilities, limitations, and near-term potential of each.

Programming Languages (PL)

- PL1. Overview of programming languages [core]
- PL2. Virtual machines [core]
- PL3. Introduction to language translation [core]
- PL4. Declarations and types [core]
- PL5. Abstraction mechanisms [core]
- PL6. Object-oriented programming [core]
- PL7. Functional programming [elective]
- PL8. Language translation systems [elective]
- PL9. Type systems [elective]
- PL10. Programming language semantics [elective]
- PL11. Programming language design [elective]

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.

PL1. Overview of programming languages [core]

Minimum core 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. Distinguish between programming-in-the-small and programming-in-the-large.

PL2. Virtual machines [core]

Minimum core 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.

PL3. Introduction to language translation [core]

Minimum core coverage time: 2 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 and where these differences are evident in the translation process.

PL4. Declarations and types [core]

Minimum core coverage time: 3 hours

Topics:

- The conception of types as a set of values with 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-inthelarge.
2. Identify and describe the properties of a variable 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).

PL5. Abstraction mechanisms [core]

Minimum core 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.

PL6. Object-oriented programming [core]

Minimum core coverage time: 10 hours

Topics:

- Object-oriented design
- Encapsulation and information-hiding
- Separation of behavior 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. Describe how the class mechanism supports encapsulation and information hiding.
4. Design, implement, and test the implementation of “is-a” relationships among objects using a class hierarchy and inheritance.
5. Compare and contrast the notions of overloading and overriding methods in an object-oriented language.
6. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
7. Describe how iterators access the elements of a container.

PL7. Functional programming [elective]

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.

PL8. Language translation systems [elective]

Topics:

- Application of regular expressions in lexical scanners
- Parsing (concrete and abstract syntax, abstract syntax trees)
- Application of context-free grammars in table-driven and recursive-descent parsing
- Symbol table management
- Code generation by tree walking
- Architecture-specific operations: instruction selection and register allocation
- Optimization techniques
- The use of tools in support of the translation process and the advantages thereof
- Program libraries and separate compilation
- Building syntax-directed tools

Learning objectives:

1. Describe the steps and algorithms used by language translators.
2. Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.
3. Discuss the effectiveness of optimization.
4. Explain the impact of a separate compilation facility and the existence of program libraries on the compilation process.

PL9. Type systems [elective]

Topics:

- Data type as set of values with set of operations
- Data types
- Elementary types
 - Product and coproduct types
 - Algebraic types
 - Recursive types
 - Arrow (function) types
 - Parameterized types
 - Type-checking models
- Semantic models of user-defined types
 - Type abbreviations
 - Abstract data types
 - Type equality

- Parametric polymorphism
- Subtype polymorphism
- Type-checking algorithms

Learning objectives:

1. Formalize the notion of typing.
2. Describe each of the elementary data types.
3. Explain the concept of an abstract data type.
4. Recognize the importance of typing for abstraction and safety.
5. Differentiate between static and dynamic typing.
6. Differentiate between type declarations and type inference.
7. Evaluate languages with regard to typing.

PL10. Programming language semantics [elective]

Topics:

- Informal semantics
- Overview of formal semantics
- Denotational semantics
- Axiomatic semantics
- Operational semantics

Learning objectives:

1. Explain the importance of formal semantics.
2. Differentiate between formal and informal semantics.
3. Describe the different approaches to formal semantics.
4. Evaluate the different approaches to formal semantics.

PL11. Programming language design [elective]

Topics:

- General principles of language design
- Design goals
- Typing regimes
- Data structure models
- Control structure models
- Abstraction mechanisms

Learning objectives:

1. Evaluate the impact of different typing regimes on language design, language usage, and the translation process.
2. Explain the role of different abstraction mechanisms in the creation of user-defined facilities.

Human-Computer Interaction (HC)

- HC1. Foundations of human-computer interaction [core]
- HC2. Building a simple graphical user interface [core]
- HC3. Human-centered software evaluation [elective]
- HC4. Human-centered software development [elective]
- HC5. Graphical user-interface design [elective]
- HC6. Graphical user-interface programming [elective]
- HC7. HCI aspects of multimedia systems [elective]
- HC8. HCI aspects of collaboration and communication [elective]

This list of topics is intended as an introduction to human-computer interaction for computer science majors. Emphasis will be placed 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. Units HC1 (Foundations of Human-Computer Interaction) and HC2 (Building a simple graphical user interface) will be required for all majors, possibly as modules in the introductory courses. The remaining units will most likely be integrated into one or two elective courses at the junior or senior level.

HC1. Foundations of human-computer interaction [core]

Minimum core coverage time: 6 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 a given icon, symbol, word, or color can have in (a) two different human cultures and (b) in a culture and one of its subcultures.
6. In what ways might the design of a computer system or application succeed or fail in terms of respecting human diversity.
7. Create and conduct a simple usability test for an existing software application.

HC2. Building a simple graphical user interface [core]

Minimum core coverage time: 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. Illustrate the effect of fundamental design principles on the structure of a graphical user interface.
4. Conduct a simple usability test for each instance and compare the results.

HC3. Human-centered software evaluation [elective]

Topics:

Setting goals for evaluation

- Evaluation without users: walkthroughs, KLM, guidelines, and standards
- Evaluation with users: usability testing, interview, survey, experiment

Learning objectives:

1. Discuss evaluation criteria: learning, task time and completion, acceptability.
2. Conduct a walkthrough and a Keystroke Level Model (KLM) analysis.
3. Summarize the major guidelines and standards.
4. Conduct a usability test, an interview, and a survey.
5. Compare a usability test to a controlled experiment.
6. Evaluate an existing interactive system with human-centered criteria and a usability test.

HC4. Human-centered software development [elective]

Topics:

- Approaches, characteristics, and overview of process
- Functionality and usability: task analysis, interviews, surveys
- Specifying interaction and presentation
- Prototyping techniques and tools
 - Paper storyboards
 - Inheritance and dynamic dispatch
 - Prototyping languages and GUI builders

Learning objectives:

1. Explain the basic types and features of human-centered development.
2. Compare human-centered development to traditional software engineering methods.
3. State three functional requirements and three usability requirements.
4. Specify an interactive object with transition networks, OO design, or scenario descriptions.
5. Discuss the pros and cons of development with paper and software prototypes.

HC5. Graphical user-interface design [elective]

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 illustrating knowledge of the concepts taught in HC3 and HC4.
4. Discuss the challenges that exist in moving from 2D to 3D interaction.

HC6. Graphical user-interface programming [elective]

Topics:

UIMS, dialogue independence and levels of analysis, Seeheim model

- Widget classes
- Event management and user interaction
- Geometry management
- GUI builders and UI programming environments
- Cross-platform design

Learning objectives:

1. Differentiate between the responsibilities of the UIMS and the application.
2. Differentiate between kernel-based and client-server models for the UI.
3. Compare the event-driven paradigm with more traditional procedural control for the UI.
4. Describe aggregation of widgets and constraint-based geometry management.
5. Explain callbacks and their role in GUI builders.
6. Identify at least three differences common in cross-platform UI design.
7. Identify as many commonalities as you can that are found in UIs across different platforms.

HC7. HCI aspects of multimedia systems [elective]

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 illustrating knowledge of the concepts taught in HC4, HC5, and HC7.

HC8. HCI aspects of collaboration and communication [elective]

Topics:

- Groupware to support specialized tasks: document preparation, multi-player games
- Asynchronous group communication: e-mail, bulletin boards
- Synchronous group communication: chat rooms, conferencing
- Online communities: MUDs/MOOs
- Software characters and intelligent agents

Learning objectives:

1. Compare the HCI issues in individual interaction with group interaction.
2. Discuss several issues of social concern raised by collaborative software.
3. Discuss the HCI issues in software that embodies human intention.
4. Describe the difference between synchronous and asynchronous communication.
5. Design, prototype, and evaluate a simple groupware or group communication application illustrating knowledge of the concepts taught in HC4, HC5, and HC8.
6. Participate in a team project for which some interaction is face-to-face and other interaction occurs via a mediating software environment.
7. Describe the similarities and differences between face-to-face and software-mediated collaboration.

Graphics and Visual Computing (GV)

- GV1. Fundamental techniques in graphics [core]
- GV2. Graphic systems [core]
- GV3. Graphic communication [elective]
- GV4. Geometric modeling [elective]
- GV5. Basic rendering [elective]
- GV6. Advanced rendering [elective]
- GV7. Advanced techniques [elective]
- GV8. Computer animation [elective]
- GV9. Visualization [elective]
- GV10. Virtual reality [elective]
- GV11. Computer vision [elective]

The area encompassed by Graphics and Visual Computing (GV) is divided into four interrelated fields:

- *Computer graphics.* Computer graphics is the art and science of communicating information using images that are generated and presented through computation. This requires (a) the design and construction of models that represent information in ways that support the creation and viewing of images, (b) the design of devices and techniques through which the person may interact with the model or the view, (c) the creation of techniques for rendering the model, and (d) 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.
- *Visualization.* The field of visualization seeks to determine and present underlying correlated structures and relationships in both scientific (computational and medical sciences) and more abstract datasets. The prime objective of the presentation should be to communicate the information in a dataset so as to enhance understanding. Although current techniques of visualization exploit visual abilities of humans, other sensory modalities, including sound and haptics (touch), are also being considered to aid the discovery process of information.
- *Virtual reality.* Virtual reality (VR) enables users to experience a three-dimensional environment generated using computer graphics, and perhaps other sensory modalities, to provide an environment for enhanced interaction between a human user and a computer-created world.
- *Computer vision.* The goal of computer vision (CV) is to deduce the properties and structure of the three-dimensional world from one or more two-dimensional images. The understanding and practice of computer vision depends upon core concepts in computing, but also relates strongly to the disciplines of physics, mathematics, and psychology.

GV1. Fundamental techniques in graphics [core]

Minimum core coverage time: 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 coordinate system and the changes required to extend 2D transformation operations to handle transformations in 3D

GV2. Graphic systems [core]

Minimum core coverage time: 1 hour

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 architectures for given applications.
2. Explain the function of various input devices.
3. Compare and contrast the techniques of raster graphics and vector graphics.
4. Use current hardware and software for creating and displaying graphics.
5. Discuss the expanded capabilities of emerging hardware and software for creating and displaying graphics.

GV3. Graphic communication [elective]

Topics:

- Psychodynamics of color and interactions among colors
- Modifications of color for vision deficiency
- Cultural meaning of different colors
- Use of effective pseudo-color palettes for images for specific audiences
- Structuring a view for effective understanding
- Image modifications for effective video and hardcopy
- Use of legends to key information to color or other visual data
- Use of text in images to present context and background information
- Visual user feedback on graphical operations

Learning objectives:

1. Explain the value of using colors and pseudo-colors.
2. Demonstrate the ability to create effective video and hardcopy images.
3. Identify effective and ineffective examples of communication using graphics.
4. Create effective examples of graphic communication, making appropriate use of color, legends, text, and/or video.

5. Create two effective examples that communicate the same content: one designed for hardcopy presentation and the other designed for online presentation.
6. Discuss the differences in design criteria for hardcopy and online presentations.

GV4. Geometric modeling [elective]

Topics:

- Polygonal representation of 3D objects
- Parametric polynomial curves and surfaces
- Constructive Solid Geometry (CSG) representation
- Implicit representation of curves and surfaces
- Spatial subdivision techniques
- Procedural models
- Deformable models
- Subdivision surfaces
- Multiresolution modeling
- Reconstruction

Learning objectives:

1. Create simple polyhedral models by surface tessellation.
2. Construct CSG models from simple primitives, such as cubes and quadric surfaces.
3. Generate a mesh representation from an implicit surface.
4. Generate a fractal model or terrain using a procedural method.
5. Generate a mesh from data points acquired with a laser scanner.

GV5. Basic rendering [elective]

Topics:

- Line generation algorithms (Bresenham)
- Font generation: outline vs. bitmap
- Light-source and material properties
- Ambient, diffuse, and specular reflections
- Phong reflection model
- Rendering of a polygonal surface; flat, Gouraud, and Phong shading
- Texture mapping, bump texture, environment map
- Introduction to ray tracing
- Image synthesis, sampling techniques, and anti-aliasing

Learning objectives:

1. Explain the operation of the Bresenham algorithm for rendering a line on a pixelbased display.
2. Explain the concept and applications of each of these techniques.
3. Demonstrate each of these techniques by creating an image using a standard API.
4. Describe how a graphic image has been created.

GV6. Advanced rendering [elective]

Topics:

- Transport equations
- Ray tracing algorithms
- Photon tracing

- Radiosity for global illumination computation, form factors
- Efficient approaches to global illumination
- Monte Carlo methods for global illumination
- Image-based rendering, panorama viewing, plenoptic function modeling
- Rendering of complex natural phenomenon
- Non-photorealistic rendering

Learning objectives:

1. Describe several transport equations in detail, noting all comprehensive effects.
2. Describe efficient algorithms to compute radiosity and explain the tradeoffs of accuracy and algorithmic performance.
3. Describe the impact of meshing schemes.
4. Explain image-based rendering techniques, light fields, and associated topics.

GV7. Advanced techniques [elective]

Topics:

- Color quantization
- Scan conversion of 2D primitive, forward differencing
- Tessellation of curved surfaces
- Hidden surface removal methods
- Z-buffer and frame buffer, color channels (a channel for opacity)

Advanced geometric modeling techniques

Learning objectives:

1. Describe the techniques identified in this section.
2. Explain how to recognize the graphics techniques used to create a particular image.
3. Implement any of the specified graphics techniques using a primitive graphics system at the individual pixel level.
4. Use common animation software to construct simple organic forms using metaball and skeleton.

GV8. Computer animation [elective]

Topics:

- Key-frame animation
- Camera animation
- Scripting system
- Animation of articulated structures: inverse kinematics
- Motion capture
- Procedural animation
- Deformation

Learning objectives:

1. Explain the spline interpolation method for producing in-between positions and orientations.
2. Compare and contrast several technologies for motion capture.
3. Use the particle function in common animation software to generate a simple animation, such as fireworks.
4. Use free-form deformation techniques to create various deformations.

GV9. Visualization [elective]

Topics:

- Basic viewing and interrogation functions for visualization
- Visualization of vector fields, tensors, and flow data
- Visualization of scalar field or height field: isosurface by the marching cube method
- Direct volume data rendering: ray-casting, transfer functions, segmentation, hardware
- Information visualization: projection and parallel-coordinates methods

Learning objectives:

1. Describe the basic algorithms behind scalar and vector visualization.
2. Describe the tradeoffs of the algorithms in terms of accuracy and performance.
3. Employ suitable theory from signal processing and numerical analysis to explain the effects of visualization operations.
4. Describe the impact of presentation and user interaction on exploration.

GV10. Virtual reality [elective]

Topics:

- Stereoscopic display
- Force feedback simulation, haptic devices
- Viewer tracking
- Collision detection
- Visibility computation
- Time-critical rendering, multiple levels of details (LOD)
- Image-base VR system
- Distributed VR, collaboration over computer network
- Interactive modeling
- User interface issues
- Applications in medicine, simulation, and training

Learning objectives:

1. Describe the optical model realized by a computer graphics system to synthesize stereoscopic view.
2. Describe the principles of different viewer tracking technologies.
3. Explain the principles of efficient collision detection algorithms for convex polyhedra.
4. Describe the differences between geometry- and image-based virtual reality.
5. Describe the issues of user action synchronization and data consistency in a networked environment.
6. Determine the basic requirements on interface, hardware, and software configurations of a VR system for a specified application.

GV11. Computer vision [elective]

Topics:

- Image acquisition
- The digital image and its properties
- Image preprocessing
- Segmentation (thresholding, edge- and region-based segmentation)
- Shape representation and object recognition
- Motion analysis

- Case studies (object recognition, object tracking)

Learning objectives:

1. Explain the image formation process.
2. Explain the advantages of two and more cameras, stereo vision.
3. Explain various segmentation approaches, along with their characteristics, differences, strengths, and weaknesses.
4. Describe object recognition based on contour- and region-based shape representations.
5. Explain differential motion analysis methods.
6. Describe the differences in object tracking methods.

Intelligent Systems (IS)

- IS1. Fundamental issues in intelligent systems [core]
- IS2. Search and constraint satisfaction [core]
- IS3. Knowledge representation and reasoning [core]
- IS4. Advanced search [elective]
- IS5. Advanced knowledge representation and reasoning [elective]
- IS6. Agents [elective]
- IS7. Natural language processing [elective]
- IS8. Machine learning and neural networks [elective]
- IS9. AI planning systems [elective]
- IS10. Robotics [elective]

The field of artificial intelligence (AI) is concerned with the design and analysis of autonomous agents. These are software systems and/or physical machines, with sensors and actuators, embodied for example within a robot or an autonomous spacecraft. An intelligent system has to perceive its environment, to act rationally towards its assigned tasks, to interact with other agents and with human beings. These capabilities are covered by topics such as computer vision, planning and acting, robotics, multiagents systems, speech recognition, and natural language understanding. They rely on a broad set of general and specialized knowledge representations and reasoning mechanisms, on problem solving and search algorithms, and on machine learning techniques.

Furthermore, artificial intelligence provides a set of tools for solving problems that are difficult or impractical to solve with other methods. These include heuristic search and planning algorithms, formalisms for knowledge representation and reasoning, machine learning techniques, and methods applicable to sensing and action problems such as speech and language understanding, computer vision, and robotics, among others. The student needs to be able to determine when an AI approach is appropriate for a given problem, and to be able to select and implement a suitable AI method.

IS1. Fundamental issues in intelligent systems [core]

Minimum core coverage time: 1 hour

Topics:

History of artificial intelligence

Philosophical questions

- The Turing test
- Searle's "Chinese Room" thought experiment
- Ethical issues in AI

Fundamental definitions

- Optimal vs. human-like reasoning
 - Optimal vs. human-like behavior
- Philosophical questions
Modeling the world
The role of heuristics

Learning objectives:

1. Describe the Turing test and the "Chinese Room" thought experiment.

2. Differentiate the concepts of optimal reasoning and human-like reasoning.
3. Differentiate the concepts of optimal behavior and human-like behavior.
4. List examples of intelligent systems that depend on models of the world.
5. Describe the role of heuristics and the need for tradeoffs between optimality and efficiency.

IS2. Search and constraint satisfaction [core]

Minimum core coverage time: 5 hours

Topics:

Problem spaces

- Brute-force search (breadth-first, depth-first, depth-first with iterative deepening)
- Best-first search (generic best-first, Dijkstra's algorithm, A*, admissibility of A*)
- Two-player games (minimax search, alpha-beta pruning)
- Constraint satisfaction (backtracking and local search methods)

Learning objectives:

1. Formulate an efficient problem space for a problem expressed in English by expressing that problem space in terms of states, operators, an initial state, and a description of a goal state.
2. Describe the problem of combinatorial explosion and its consequences.
3. Select an appropriate brute-force search algorithm for a problem, implement it, and characterize its time and space complexities.
4. Select an appropriate heuristic search algorithm for a problem and implement it by designing the necessary heuristic evaluation function.
5. Describe under what conditions heuristic algorithms guarantee optimal solution.
6. Implement minimax search with alpha-beta pruning for some two-player game.
7. Formulate a problem specified in English as a constraint-satisfaction problem and implement it using a chronological backtracking algorithm.

IS3. Knowledge representation and reasoning [core]

Minimum core coverage time: 4 hours

Topics:

- Review of propositional and predicate logic
- Resolution and theorem proving
- Nonmonotonic inference
- Probabilistic reasoning
- Bayes theorem

Learning objectives:

1. Explain the operation of the resolution technique for theorem proving.
2. Explain the distinction between monotonic and nonmonotonic inference.
3. Discuss the advantages and shortcomings of probabilistic reasoning.
4. Apply Bayes theorem to determine conditional probabilities.

IS4. Advanced search [elective]

Topics:

- Genetic algorithms
- Simulated annealing

- Local search

Learning objectives:

1. Explain what genetic algorithms are and contrast their effectiveness with the classic problem-solving and search techniques.
2. Explain how simulated annealing can be used to reduce search complexity and contrast its operation with classic search techniques.
3. Apply local search techniques to a classic domain.

IS5. Advanced knowledge representation and reasoning [elective]

Topics:

Structured representation

- Frames and objects
- Description logics
- Inheritance systems

Nonmonotonic reasoning

- Nonclassical logics
- Default reasoning
- Belief revision
- Preference logics
- Integration of knowledge sources
- Aggregation of conflicting belief

Reasoning on action and change

- Situation calculus
- Event calculus
- Ramification problems Temporal and spatial reasoning Uncertainty
- Probabilistic reasoning
- Bayesian nets
- Fuzzy sets and possibility theory
- Decision theory

Knowledge representation for diagnosis, qualitative representation

Learning objectives:

1. Compare and contrast the most common models used for structured knowledge representation, highlighting their strengths and weaknesses.
2. Characterize the components of nonmonotonic reasoning and its usefulness as a representational mechanisms for belief systems.
3. Apply situation and event calculus to problems of action and change.
4. Articulate the distinction between temporal and spatial reasoning, explaining how they interrelate.
5. Describe and contrast the basic techniques for representing uncertainty.
6. Describe and contrast the basic techniques for diagnosis and qualitative representation.

IS6. Agents [elective]

Topics:

Definition of agents

Successful applications and state-of-the-art agent-based systems

Agent architectures

- Simple reactive agents
- Reactive planners
- Layered architectures
- Example architectures and applications

Agent theory

- Commitments
- Intentions
- Decision-theoretic agents
- Markov decision processes (MDP)

Software agents, personal assistants, and information access

- Collaborative agents
- Information-gathering agents
 - Believable agents (synthetic characters, modeling emotions in agents)
 - Learning agents
 - Multi-agent systems
- Economically inspired multi-agent systems
- Collaborating agents
- Agent teams
- Agent modeling
- Multi-agent learning Introduction to robotic agents Mobile agents

Learning objectives:

1. Explain how an agent differs from other categories of intelligent systems.
2. Characterize and contrast the standard agent architectures.
3. Describe the applications of agent theory, to domains such as software agents, personal assistants, and believable agents.
4. Describe the distinction between agents that learn and those that don't.
5. Demonstrate using appropriate examples how multi-agent systems support agent interaction.
6. Describe and contrast robotic and mobile agents.

IS7. Natural language processing [elective]

Topics:

- Deterministic and stochastic grammars
- Parsing algorithms
- Corpus-based methods
- Information retrieval
- Language translation
- Speech recognition

Learning objectives:

1. Define and contrast deterministic and stochastic grammars, providing examples to show the adequacy of each.

2. Identify the classic parsing algorithms for parsing natural language.
3. Defend the need for an established corpus.
4. Give examples of catalog and look up procedures in a corpus-based approach.
5. Articulate the distinction between techniques for information retrieval, language translation, and speech recognition.

IS8. Machine learning and neural networks [elective]

Topics:

- Definition and examples of machine learning
- Supervised learning
- Learning decision trees
- Learning neural networks
- Learning belief networks
- The nearest neighbor algorithm
- Learning theory
- The problem of overfitting
- Unsupervised learning
- Reinforcement learning

Learning objectives:

1. Explain the differences among the three main styles of learning: supervised, reinforcement, and unsupervised.
2. Implement simple algorithms for supervised learning, reinforcement learning, and unsupervised learning.
3. Determine which of the three learning styles is appropriate to a particular problem domain.
4. Compare and contrast each of the following techniques, providing examples of when each strategy is superior: decision trees, neural networks, and belief networks..
5. Implement a simple learning system using decision trees, neural networks and/or belief networks, as appropriate.
6. Characterize the state of the art in learning theory, including its achievements and its shortcomings.
7. Explain the nearest neighbor algorithm and its place within learning theory.
8. Explain the problem of overfitting, along with techniques for detecting and managing the problem.

IS9. AI planning systems [elective]

Topics:

- Definition and examples of planning systems
- Planning as search
- Operator-based planning
- Propositional planning
- Extending planning systems (case-based, learning, and probabilistic systems)
- Static world planning systems
- Planning and execution
- Planning and robotics

Learning objectives:

1. Define the concept of a planning system.
2. Explain how planning systems differ from classical search techniques.
3. Articulate the differences between planning as search, operator-based planning, and propositional planning, providing examples of domains where each is most applicable.
4. Define and provide examples for each of the following techniques: case-based, learning, and probabilistic planning.
5. Compare and contrast static world planning systems with those need dynamic execution.
6. Explain the impact of dynamic planning on robotics.

IS10. Robotics [elective]

Topics:

Overview

- State-of-the-art robot systems
- Planning vs. reactive control
- Uncertainty in control
- Sensing
- World models
 - Configuration space
 - Planning
 - Sensing
 - Robot programming
 - Navigation and control

Learning objectives:

1. Outline the potential and limitations of today's state-of-the-art robot systems.
2. Implement configuration space algorithms for a 2D robot and complex polygons.
3. Implement simple motion planning algorithms.
4. Explain the uncertainties associated with sensors and how to deal with those uncertainties.
5. Design a simple control architecture.
6. Describe various strategies for navigation in unknown environments, including the strengths and shortcomings of each.
7. Describe various strategies for navigation with the aid of landmarks, including the strengths and shortcomings of each.

Information Management (IM)

- IM1. Information models and systems [core]
- IM2. Database systems [core]
- IM3. Data modeling [core]
- IM4. Relational databases [elective]
- IM5. Database query languages [elective]
- IM6. Relational database design [elective]
- IM7. Transaction processing [elective]
- IM8. Distributed databases [elective]
- IM9. Physical database design [elective]
- IM10. Data mining [elective]
- IM11. Information storage and retrieval [elective]
- IM12. Hypertext and hypermedia [elective]
- IM13. Multimedia information and systems [elective]
- IM14. Digital libraries [elective]

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.

IM1. Information models and systems [core]

Minimum core coverage time: 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.

IM2. Database systems [core]

Minimum core coverage time: 3 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.

IM3. Data modeling [core]

Minimum core coverage time: 4 hours

Topics:

Data modeling

- Conceptual models (including entity-relationship and UML)
- Object-oriented model
- Relational data model

Learning objectives:

1. Categorize data models based on the types of concepts that they provide to describe the database structure—that is, conceptual data model, physical data model, and representational data model.
2. Describe the modeling concepts and notation of the entity-relationship model and UML, including their use in data modeling.
3. Describe the main concepts of the OO model such as object identity, type constructors, encapsulation, inheritance, polymorphism, and versioning.
4. Define the fundamental terminology used in the relational data model .
5. Describe the basic principles of the relational data model.
6. Illustrate the modeling concepts and notation of the relational data model.

IM4. Relational databases [elective]

Topics:

- Mapping conceptual schema to a relational schema
- Entity and referential integrity
- Relational algebra and relational calculus

Learning objectives:

1. Prepare a relational schema from a conceptual model developed using the entityrelationship model
2. Explain and demonstrate the concepts of entity integrity constraint and referential integrity constraint (including definition of the concept of a foreign key).
3. *Demonstrate use of the relational algebra operations from mathematical set theory (union, intersection, difference, and cartesian product) and the relational algebra operations developed specifically for relational databases (select, product, join, and division).*
4. Demonstrate queries in the relational algebra.
5. Demonstrate queries in the tuple relational calculus.

IM5. Database query languages [elective]

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).

IM6. Relational database design [elective]

Topics:

- Database design
- Functional dependency
- Normal forms (1NF, 2NF, 3NF, BCNF)
- Multivalued dependency (4NF)
- Join dependency (PJNF, 5NF)
- Representation theory

Learning objectives:

1. Determine the functional dependency between two or more attributes that are a subset of a relation.
2. Describe what is meant by 1NF, 2NF, 3NF, and BCNF.
3. Identify whether a relation is in 1NF, 2NF, 3NF, or BCNF.
4. Normalize a 1NF relation into a set of 3NF (or BCNF) relations and denormalize a relational schema.

5. Explain the impact of normalization on the efficiency of database operations, especially query optimization.
6. Describe what is a multivalued dependency and what type of constraints it specifies.
7. Explain why 4NF is useful in schema design.

IM7. Transaction processing [elective]

Topics:

- Transactions
- Failure and recovery
- Concurrency control

Learning objectives:

1. Create a transaction by embedding SQL into an application program.
2. Explain the concept of implicit commits.
3. Describe the issues specific to efficient transaction execution.
4. Explain when and why rollback is needed and how logging assures proper rollback.
5. Explain the effect of different isolation levels on the concurrency control mechanisms.
6. Choose the proper isolation level for implementing a specified transaction protocol.

IM8. Distributed databases [elective]

Topics:

- Distributed data storage
- Distributed query processing
- Distributed transaction model
- Concurrency control
- Homogeneous and heterogeneous solutions
- Client-server

Learning objectives:

1. Explain the techniques used for data fragmentation, replication, and allocation during the distributed database design process.
2. Evaluate simple strategies for executing a distributed query to select the strategy that minimizes the amount of data transfer.
3. Explain how the two-phase commit protocol is used to deal with committing a transaction that accesses databases stored on multiple nodes.
4. Describe distributed concurrency control based on the distinguished copy techniques and the voting method.
5. Describe the three levels of software in the client-server model.

IM9. Physical database design [elective]

Topics:

- Storage and file structure
- Indexed files
- Hashed files
- Signature files
- B-trees
- Files with dense index
- Files with variable length records

- Database efficiency and tuning

Learning objectives:

1. Explain the concepts of records, record types, and files, as well as the different techniques for placing file records on disk.
2. Give examples of the application of primary, secondary, and clustering indexes.
3. Distinguish between a nondense index and a dense index.
4. Implement dynamic multilevel indexes using B-trees.
5. Explain the theory and application of internal and external hashing techniques.
6. Use hashing to facilitate dynamic file expansion.
7. Describe the relationships among hashing, compression, and efficient database searches.
8. Evaluate costs and benefits of various hashing schemes.
9. Explain how physical database design affects database transaction efficiency.

IM10. Data mining [elective]

Topics:

- The usefulness of data mining
- Associative and sequential patterns
- Data clustering
- Market basket analysis
- Data cleaning
- Data visualization

Learning objectives:

1. Compare and contrast different conceptions of data mining as evidenced in both research and application.
2. Explain the role of finding associations in commercial market basket data.
3. Characterize the kinds of patterns that can be discovered by association rule mining.
4. Describe how to extend a relational system to find patterns using association rules.
5. Evaluate methodological issues underlying the effective application of data mining.
6. Identify and characterize sources of noise, redundancy, and outliers in presented data.
7. Identify mechanisms (on-line aggregation, anytime behavior, interactive visualization) to close the loop in the data mining process.
8. Describe why the various close-the-loop processes improve the effectiveness of data mining.

IM11. Information storage and retrieval [elective]

Topics:

Characters, strings, coding, text

- Documents, electronic publishing, markup, and markup languages
- Tries, inverted files, PAT trees, signature files, indexing
- Morphological analysis, stemming, phrases, stop lists
- Term frequency distributions, uncertainty, fuzziness, weighting
- Vector space, probabilistic, logical, and advanced models
- Information needs, relevance, evaluation, effectiveness
- Thesauri, ontologies, classification and categorization, metadata
- Bibliographic information, bibliometrics, citations
- Routing and (community) filtering

- Search and search strategy, information seeking behavior, user modeling, feedback
- Information summarization and visualization
- Integration of citation, keyword, classification scheme, and other terms
- Protocols and systems (including Z39.50, OPACs, WWW engines, research systems)

Learning objectives:

1. Explain basic information storage and retrieval concepts.
2. Describe what issues are specific to efficient information retrieval.
3. Give applications of alternative search strategies and explain why the particular search strategy is appropriate for the application.
4. Perform Internet-based research.
5. Design and implement a small to medium size information storage and retrieval system.

IM12. Hypertext and hypermedia [elective]

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.

IM13. Multimedia information and systems [elective]

Topics:

- Devices, device drivers, control signals and protocols, DSPs
- Applications, media editors, authoring systems, and authoring
- Streams/structures, capture/represent/transform, spaces/domains, compression/coding
- Content-based analysis, indexing, and retrieval of audio, images, and video
- Presentation, rendering, synchronization, multi-modal integration/interfaces
- Real-time delivery, quality of service, audio/video conferencing, video-on-demand

Learning objectives:

1. Describe the media and supporting devices commonly associated with multimedia information and systems.
2. Explain basic multimedia presentation concepts.

3. Demonstrate the use of content-based information analysis in a multimedia information system.
4. Critique multimedia presentations in terms of their appropriate use of audio, video, graphics, color, and other information presentation concepts.
5. Implement a multimedia application using a commercial authoring system.

IM14. Digital libraries [elective]

Topics:

- Digitization, storage, and interchange
- Digital objects, composites, and packages
- Metadata, cataloging, author submission
- Naming, repositories, archives
- Spaces (conceptual, geographical, 2/3D, VR)
- Architectures (agents, buses, wrappers/mediators), interoperability
- Services (searching, linking, browsing, and so forth)
- Intellectual property rights management, privacy, protection (watermarking)

Archiving and preservation, integrity

Learning objectives:

1. Explain the underlying technical concepts in building a digital library.
2. Describe the basic service requirements for searching, linking, and browsing.
3. Critique scenarios involving appropriate and inappropriate use of a digital library, and determine the social, legal, and economic consequences for each scenario.
4. Describe some of the technical solutions to the problems related to archiving and preserving information in a digital library.
5. Design and implement a small digital library.

Social and Professional Issues (SP)

- SP1. History of computing [core]
- SP2. Social context of computing [core]
- SP3. Methods and tools of analysis [core]
- SP4. Professional and ethical responsibilities [core]
- SP5. Risks and liabilities of computer-based systems [core]
- SP6. Intellectual property [core]
- SP7. Privacy and civil liberties [core]
- SP8. Computer crime [elective]
- SP9. Economic issues in computing [elective]
- SP10. Philosophical frameworks [elective]

Although technical issues are obviously central to any computing curriculum, they do not by themselves constitute a complete educational program in the field. Students must also develop an understanding of the social and professional context in which computing is done.

This need to incorporate the study of social issues into the curriculum was recognized in the following excerpt from *Computing Curricula 1991* [Tucker91]:

Undergraduates also need to understand the basic cultural, social, legal, and ethical issues inherent in the discipline of computing. They should understand where the discipline has been, where it is, and where it is heading. They should also understand their individual roles in this process, as well as appreciate the philosophical questions, technical problems, and aesthetic values that play an important part in the development of the discipline.

Students also need to develop the ability to ask serious questions about the social impact of computing and to evaluate proposed answers to those questions.

Future practitioners must be able to anticipate the impact of introducing a given product into a given environment. Will that product enhance or degrade the quality of life? What will the impact be upon individuals, groups, and institutions?

Finally, students need to be aware of the basic legal rights of software and hardware vendors and users, and they also need to appreciate the ethical values that are the basis for those rights. Future practitioners must understand the responsibility that they will bear, and the possible consequences of failure. They must understand their own limitations as well as the limitations of their tools. All practitioners must make a long-term commitment to remaining current in their chosen specialties and in the discipline of computing as a whole. The material in this knowledge area is best covered through a combination of one required course along with short modules in other courses. On the one hand, some units listed as core—in particular, SP2, SP3, SP4, and SP6—do not readily lend themselves to being covered in other traditional courses. Without a standalone course, it is difficult to cover these topics appropriately. On the other hand, if ethical considerations are covered only in the standalone course and not “in context,” it will reinforce the false notion that technical processes are void of ethical issues. Thus it is important that several traditional courses include modules that analyze ethical considerations in the context of the technical subject matter of the course. Courses in areas such as software engineering, databases, computer networks, and introduction to computing provide obvious context for analysis of ethical issues. However, an ethics-related module could be developed for almost any course in the curriculum. It would be explicitly against the spirit of the recommendations to have only a standalone course. Running through all of the issues in this

area is the need to speak to the computer practitioner's responsibility to proactively address these issues by both moral and technical actions.

The ethical issues discussed in any class should be directly related to and arise naturally from the subject matter of that class. Examples include a discussion in the database course of data aggregation or data mining, or a discussion in the software engineering course of the potential conflicts between obligations to the customer and obligations to the user and others affected by their work. Programming assignments built around applications such as controlling the movement of a laser during eye surgery can help to address the professional, ethical and social impacts of computing.

There is an unresolved pedagogical conflict between having the core course at the lower (freshman-sophomore) level versus the upper (junior-senior) level. Having the course at the lower level

1. Allows for coverage of methods and tools of analysis (SP3) prior to analyzing ethical issues in the context of different technical areas
2. Assures that students who drop out early to enter the workforce will still be introduced to some professional and ethical issues.

On the other hand, placing the course too early may lead to the following problems:

1. Lower-level students may not have the technical knowledge and intellectual maturity to support in-depth ethical analysis. Without basic understanding of technical alternatives, it is difficult to consider their ethical implications.
2. Students need a certain level of maturity and sophistication to appreciate the background and issues involved. For that reason, students should have completed at least the discrete mathematics course and the second computer science course. Also, if students take a technical writing course, it should be a prerequisite or corequisite for the required course in the SP area.
3. Some programs may wish to use the course as a "capstone" experience for seniors.

Although items SP2 and SP3 are listed with a number of hours associated, they are fundamental to all the other topics. Thus, when covering the other areas, instructors should continually be aware of the social context issues and the ethical analysis skills. In practice, this means that the topics in SP2 and SP3 will be continually reinforced as the material in the other areas is covered.

SP1. History of computing [core]

Minimum core 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 personal computers and the Internet.
3. Identify significant continuing trends in the history of the computing field.

SP2. Social context of computing [core]

Minimum core coverage time: 3 hours

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 through the use of 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.

SP3. Methods and tools of analysis [core]

Minimum core coverage time: 2 hours

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.

SP4. Professional and ethical responsibilities [core]

Minimum core coverage time: 3 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

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, 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.

SP5. Risks and liabilities of computer-based systems [core]

Minimum core coverage time: 2 hours

Topics:

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

Learning objectives:

1. Explain the limitations of testing as a means to ensure correctness.
2. Describe the differences between correctness, reliability, and safety.
3. Discuss the potential for hidden problems in reuse of existing components.
4. Describe current approaches to managing risk, and characterize the strengths and shortcomings of each.

SP6. Intellectual property [core]

Minimum core coverage time: 3 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.
2. Discuss the legal background of copyright in national and international law.
3. Explain how patent and copyright laws may vary internationally.
4. Outline the historical development of software patents.
5. Discuss the consequences of software piracy on software developers and the role of relevant enforcement organizations.

SP7. Privacy and civil liberties [core]

Minimum core coverage time: 2 hours

Topics:

- Ethical and legal basis for privacy protection
- Privacy implications of massive database systems
- Technological strategies for privacy protection
- Freedom of expression in cyberspace
- International and intercultural implications

Learning objectives:

1. Summarize the legal bases for the right to privacy and freedom of expression in one's own nation and how those concepts vary from country to country.
2. Describe current computer-based threats to privacy.

3. Explain how the Internet may change the historical balance in protecting freedom of expression.
4. Explain both the disadvantages and advantages of free expression in cyberspace.
5. Describe trends in privacy protection as exemplified in technology.

SP8. Computer crime [elective]

Topics:

- History and examples of computer crime
- “Cracking” (“hacking”) and its effects
- Viruses, worms, and Trojan horses
- Crime prevention strategies

Learning objectives:

1. Outline the technical basis of viruses and denial-of-service attacks.
2. Enumerate techniques to combat “cracker” attacks.
3. Discuss several different “cracker” approaches and motivations.
4. Identify the professional’s role in security and the tradeoffs involved.

SP9. Economic issues in computing [elective]

Topics:

Monopolies and their economic implications

- Effect of skilled labor supply and demand on the quality of computing products
- Pricing strategies in the computing domain
- Differences in access to computing resources and the possible effects thereof

Learning objectives:

1. Summarize the rationale for antimonopoly efforts.
2. Describe several ways in which the information technology industry is affected by shortages in the labor supply.
3. Suggest and defend ways to address limitations on access to computing.
4. Outline the evolution of pricing strategies for computing goods and services.

SP10. Philosophical frameworks [elective]

Topics:

- Philosophical frameworks, particularly utilitarianism and deontological theories
- Problems of ethical relativism
- Scientific ethics in historical perspective
- Differences in scientific and philosophical approaches

Learning objectives:

1. Summarize the basic concepts of relativism, utilitarianism, and deontological theories.
2. Recognize the distinction between ethical theory and professional ethics.
3. Identify the weaknesses of the “hired agent” approach, strict legalism, naïve egoism, and naïve relativism as ethical frameworks.

Software Engineering (SE)

- SE1. Software design [core]
- SE2. Using APIs [core]
- SE3. Software tools and environments [core]
- SE4. Software processes [core]
- SE5. Software requirements and specifications [core]
- SE6. Software validation [core]
- SE7. Software evolution [core]
- SE8. Software project management [core]
- SE9. Component-based computing [elective]
- SE10. Formal methods [elective]
- SE11. Software reliability [elective]
- SE12. Specialized systems development [elective]

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.

SE1. Software design [core]

Minimum core coverage time: 8 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, an accepted program design methodology (e.g., structured or object-oriented), and appropriate design notation.
6. Conduct a software design review using appropriate guidelines.
7. Evaluate a software design at the component level.
8. Evaluate a software design from the perspective of reuse.

SE2. Using APIs [core]

Minimum core coverage time: 5 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. Use class browsers and related tools during the development of applications using APIs.
3. Design, implement, test, and debug programs that use large-scale API packages.

SE3. Software tools and environments [core]

Minimum core coverage time: 3 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 tools to support the 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 capability to use a range of software tools in support of the development of a software product of medium size.

SE4. Software processes [core]

Minimum core coverage time: 2 hours

Topics:

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

Learning objectives:

1. Explain the software life cycle and its phases including the deliverables that are produced.

2. Select, with justification the software development models most appropriate for the development and maintenance of a diverse range of software products.
3. Explain the role of process maturity models.
4. Compare the traditional waterfall model to the incremental model, the object-oriented model, and other appropriate models.
5. For each of various software project scenarios, describe the project's place in the software life cycle, identify the particular tasks that should be performed next, and identify metrics appropriate to those tasks.

SE5. Software requirements and specifications [core]

Minimum core coverage time: 4 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.
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.

SE6. Software validation [core]

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

SE7. Software evolution [core]

Minimum core coverage time: 3 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. Discuss the challenges of maintaining legacy systems and the need for reverse engineering.
3. Outline the process of regression testing and its role in release management.
4. Estimate the impact of a change request to an existing product of medium size.
5. Develop a plan for re-engineering a medium-sized product in response to a change request.
6. Discuss the advantages and disadvantages of software reuse.
7. Exploit opportunities for software reuse in a given context.

SE8. Software project management [core]

Minimum core coverage time: 3 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.

SE9. Component-based computing [elective]

Topics:

Fundamentals

- The definition and nature of components
- Components and interfaces
- Interfaces as contracts
- The benefits of components

Basic techniques

- Component design and assembly
- Relationship with the client-server model and with patterns
- Use of objects and object lifecycle services
- Use of object brokers
- Marshalling
 - Applications (including the use of mobile components)
 - Architecture of component-based systems
 - Component-oriented design
 - Event handling: detection, notification, and response
 - Middleware
- The object-oriented paradigm within middleware
- Object request brokers
- Transaction processing monitors
- Workflow systems
- State-of-the-art tools

Learning objectives:

1. Explain and apply recognized principles to the building of high-quality software components.
2. Discuss and select an architecture for a component-based system suitable for a given scenario.
3. Identify the kind of event handling implemented in one or more given APIs.
4. Explain the role of objects in middleware systems and the relationship with components.
5. Apply component-oriented approaches to the design of a range of software including those required for concurrency and transactions, reliable communication services, database interaction including services for remote query and database management, secure communication and access.

SE10. Formal methods [elective]

Topics:

- Formal methods concepts
- Formal specification languages
- Executable and non-executable specifications
- Pre and post assertions
- Formal verification

Learning objectives:

1. Apply formal verification techniques to software segments with low complexity.

2. Discuss the role of formal verification techniques in the context of software validation and testing.
3. Explain the potential benefits and drawbacks of using formal specification languages.
4. Create and evaluate pre- and post-assertions for a variety of situations ranging from simple through complex.
5. Using a common formal specification language, formulate the specification of a simple software system and demonstrate the benefits from a quality perspective.

SE11. Software reliability [elective]

Topics:

- Software reliability models
- Redundancy and fault tolerance
- Defect classification
- Probabilistic methods of analysis

Learning objectives:

1. Demonstrate the ability to apply multiple methods to develop reliability estimates for a software system.
2. Identify and apply redundancy and fault tolerance for a medium-sized application.
3. Explain the problems that exist in achieving very high levels of reliability.
4. Identify methods that will lead to the realization of a software architecture that achieves a specified reliability level.

SE12. Specialized systems development [elective]

Topics:

- Real-time systems
- Client-server systems
- Distributed systems
- Parallel systems
- Web-based systems
- High-integrity systems

Learning objectives:

1. Identify and discuss different specialized systems.
2. Discuss life cycle and software process issues in the context of software systems designed for a specialized context.
3. Select, with appropriate justification, approaches that will result in the efficient and effective development and maintenance of specialized software systems.
4. Given a specific context and a set of related professional issues, discuss how a software engineer involved in the development of specialized systems should respond to those issues.
5. Outline the central technical issues associated with the implementation of specialized systems development.

Computational Science and Numerical Methods (CN)

CN1. Numerical analysis [elective]

CN2. Operations research [elective]

CN3. Modeling and simulation [elective]

CN4. High-performance computing [elective]

From the earliest days of the discipline, numerical methods and the techniques of scientific computing have constituted a major area of computer science research. As computers increase in their problem-solving power, this area—like much of the discipline—has grown in both breadth and importance. At the end of the millennium, scientific computing stands as an intellectual discipline in its own right, closely related to but nonetheless distinct from computer science.

Although courses in numerical methods and scientific computing are extremely valuable components of an undergraduate program in computer science, the CC2001 Task Force believes that none of the topics in this area represent core knowledge. From our surveys of curricula and interaction with the computer science education community, we are convinced no consensus exists that this material is essential for all CS undergraduates. It remains a vital part of the discipline, but need not be a part of every program. For those who choose to pursue it, this area offers exposure to many valuable ideas and techniques, including precision of numerical representation, error analysis, numerical techniques, parallel architectures and algorithms, modeling and simulation, and scientific visualization. At the same time, students who take courses in this area have an opportunity to apply these techniques in a wide range of application areas, such as the following:

- Molecular dynamics
- Fluid dynamics
- Celestial mechanics
- Economic forecasting
- Optimization problems
- Structural analysis of materials
- Bioinformatics
- Computational biology
- Geologic modeling
- Computerized tomography

Each of the units in this area corresponds to a full-semester course at most institutions. The level of specification of the topic descriptions and the learning objectives is therefore different from that used in other areas in which the individual units typically require smaller blocks of time.

CN1. Numerical analysis [elective]

Topics:

- Floating-point arithmetic
- Error, stability, convergence
- Taylor's series
- Iterative solutions for finding roots (Newton's Method)
- Curve fitting; function approximation
- Numerical differentiation and integration (Simpson's Rule)
- Explicit and implicit methods
- Differential equations (Euler's Method)
- Linear algebra
- Finite differences

Learning objectives:

1. Compare and contrast the numerical analysis techniques presented in this unit.
2. Define error, stability, machine precision concepts. and the inexactness of computational approximations.
3. Identify the sources of inexactness in computational approximations.
4. Design, code, test, and debug programs that implement numerical methods.

CN2. Operations research [elective]

Topics:

Linear programming

- Integer programming
- The Simplex method Probablistic modeling Queueing theory
- Petri nets
- Markov models and chains
Optimization
Network analysis and routing algorithms
Prediction and estimation
- Decision analysis
- Forecasting
- Risk management
- Econometrics, microeconomics
- Sensitivity analysis
Dynamic programming
Sample applications
Software tools

Learning objectives:

1. Apply the fundamental techniques of operations research.
2. Describe several established techniques for prediction and estimation.
3. Design, code, test, and debug application programs to solve problems in the domain of operations research.

CN3. Modeling and simulation [elective]

Topics:

Random numbers

- Pseudorandom number generation and testing
- Monte Carlo methods
- Introduction to distribution functions

Simulation modeling

- Discrete-event simulation
- Continuous simulation
Verification and validation of simulation models
- Input analysis
- Output analysis Queueing theory models Sample applications

Learning objectives:

1. Discuss the fundamental concepts of computer simulation.

2. Evaluate models for computer simulation.
3. Compare and contrast methods for random number generation.
4. Design, code, test, and debug simulation programs.

CN4. High-performance computing [elective]

Topics:

Introduction to high-performance computing

- History and importance of computational science
- Overview of application areas
- Review of required skills

High-performance computing

- Processor architectures
- Memory systems for high performance
- Input/output devices
- Pipelining
- Parallel languages and architectures

Scientific visualization

- Presentation of results
- Data formats
- Visualization tools and packages

Sample problems

- Ocean and atmosphere models
- Seismic wave propagation
- N-body systems (the Barnes-Hut algorithm)
- Chemical reactions
- Phase transitions
- Fluid flow

Learning objectives:

1. Recognize problem areas where computational modeling enhances current research methods.
2. Compare and contrast architectures for scientific and parallel computing, recognizing the strengths and weaknesses of each.
3. Implement simple performance measurements for high-performance systems.
4. Design, code, test, and debug programs using techniques of numerical analysis, computer simulation, and scientific visualization.