The Science of Computing

What is computer science?

Peter J. Denning

Because computer science affects every other scientific discipline, it seems appropriate for a journal that spans the disciplines to have a regular column devoted to examining this field and its pervasive influence. As its name suggests, the column will focus on the science of computing, rather than on the computers themselves. This first installment is somewhat longer than its successors will be, because columnist Peter Denning begins, fittingly, with a definition of computer science, and in so doing maps out territory he will cover in future columns. He turns in the next issue, for example, to networking, with a look at supercomputing and the proposed Sciencenet.

Peter Denning is director of the Research Institute for Advanced Computer Science at the NASA Ames Research Center. His doctorate, from MIT in 1968, is in electrical engineering. He is Editor-in-Chief of the monthly magazine of the Association for Computing Machinery, Communications of the ACM.

Computer science is the body of knowledge dealing with the design, analysis, implementation, efficiency, and application of processes that transform information. The fundamental question underlying all of computer science is "what can be automated?" (*1*). This discipline was born in the mid-1940s with the invention of the stored-program electronic computer and has grown rapidly ever since.

Computer science has deep roots in mathematics, engineering, and logic. For several thousand years, a principal concern of mathematics has been calculation. Many models of physical phenomena have been used to derive equations whose solutions yield predictions of those phenomena—for example, calculations of orbital trajectories, weather forecasts, and fluid flows. Many general methods for solving such equations have been devised—for example, algorithms for systems of linear equations, differential equations, and integrating functions. For almost the same period, a principal concern of engineering has been calculations that aid in the design of mechanical systems. Examples include algorithms for evaluating stresses in static objects, calculating momenta of moving objects, and measuring distances much larger or smaller than our immediate perception.

One product of the long interaction between engineering and mathematics has been mechanical aids for calculating. Some surveyors' and navigators' instruments date back a thousand years. Pascal and Leibniz built arithmetic calculators in the middle 1600s. In the 1830s, Babbage conceived of an "analytical engine" that could mechanically and without error evaluate logarithms, trigonometric functions, and other general arithmetic functions. His machine, never completed, served as an inspiration for later work. In the 1920s, Bush constructed an electronic analog computer for solving general systems of differential equations. By the 1920s electromechanical calculating machines capable of addition, subtraction, multiplication, division, and square root were available. The electronic flip-flop provided a natural bridge from these machines to digital versions with no moving parts.

Logic is also an old discipline, concerned with criteria of validity of inference and formal principles of reasoning. Since the days of Euclid, it has been a tool for rigorous mathematical and scientific argument. By 1830, it was obvious that all the known deductive systems were incomplete because paradoxes could always be found. This led to a century-long search for a "complete" deductive system—within which it would be possible to determine mechanically whether or not any given statement is either true or false. In 1931, Gödel published his "incompleteness theorem" showing that there is no such system. In the late 1930s, Turing discovered a similar result, that there are problems that cannot be solved by any mechanical procedure. The importance of logic was not only its deep insight into the limits of automatic calculation, but also its focusing on the possibility that strings of symbols, perhaps encoded as numbers, can be interpreted both as data and as programs.

This insight is the key idea that distinguishes the stored-program computer from calculating machines. The steps of the algorithm are represented as binary codes and stored in the memory for later decoding and execution by the processor. The binary code can be derived mechanically from a higher-level symbolic form, the programming language.

It is the explicit, and intricate, intertwining of the ancient threads of calculation and logical symbol manipulation that marks the birth of the discipline of computer science.

Computer science has grown from infancy in the 1940s to a broad discipline in the 1980s. The box below traces this development, showing times at which new subfields made the transition from poorly understood sets of techniques to well-understood sets of core principles. The dates shown in the box are, of course, approximate. They represent my estimates of when these areas were included in the required courses in a significant number of computer science departments. Artificial intelligence, which has been an active research area since the early days of computer science, is now making the transition from elective to required status at many universities.

The eleven areas are by no means mutually exclusive. Each has its own theoretical component; most have devised specialized programming languages as notation for algorithms and data structures; most implementa-

The evolution of computer science

Theory	1940
Numerical computation	1945
Architecture	1950
Programming languages and methodology	1960
Algorithms and data structures	1968
Operating systems	1971
Networks	1975
Human interface	1978
Database systems	1980
Concurrent computation	1982
Artificial intelligence	1986 (?)

tions are on machines with operating systems connected to networks; most deal with problems having components that can execute in parallel. Some subdisciplines, such as software engineering, embrace all eleven areas.

The following paragraphs outline the principal content of the eleven areas, listing the fundamental questions and the major accomplishments of each.

THEORY. This area deals with the basic mathematics underlying computation. The fundamental questions are: What problems can machines solve? What are optimal algorithms for given classes of problems? What is the intrinsic best- and worst-case performance of given classes of machines for given classes of problems? What problems are equivalent to each other in computational difficulty? The major accomplishments are:

1. Computability theory, which defines what machines can and cannot do. Branches include automata and formal language theory.
2. Complexity theory, which tells how to measure the time and space requirements of computable functions. This theory relates a problem's size with the best- or worst-case performance of algorithms that solve that problem.
3. Classification of problems into complexity classes, such as those solvable deterministically in polynomially bounded time (P-problems) and those solvable nondeterministically in polynomially bounded time (NP-problems).
4. Automatic theorem proving.

NUMERICAL COMPUTATION. This area deals with general methods of efficiently and accurately solving equations resulting from mathematical models of systems. The fundamental questions are: How can we accurately approximate continuous or infinite processes by finite discrete processes? How do we cope with the errors arising from these approximations? How rapidly can a given class of equations be solved for a given level of accuracy? How can symbolic manipulations on equations, such as integration, differentiation, or reduction to minimal terms, be carried out? How can the answers to these questions be incorporated into efficient, reliable, high-quality mathematical software packages? The major accomplishments are:

1. Theories of stability of methods and error propagation resulting from finite and discrete representations—in particular, backward error analysis.
2. Fast algorithms for certain problems such as the fast Fourier transform and solution of Poisson's equation. Extensive assessment of algorithms for accuracy and efficiency.
3. The finite element model for a large class of problems specifiable by regular meshes and boundary values. Associated iterative methods and convergence theory. Automatic grid refinement during numerical integration.
4. Mathematical software packages for handling general problems involving matrices, ordinary differential equations, and statistics; and less general problems involving partial differential equations, optimizations, and nonlinear equations.
5. Symbolic manipulators capable of powerful and nonobvious reductions, differentiations, and integrations of expressions.

ARCHITECTURE. This area deals with methods of organizing many hardware and software components into efficient, reliable systems. The fundamental questions are: What are the best methods of implementing processing, memory, and communication functions in a machine? How do we build large computational systems in such a way that we can convincingly demonstrate that they work as intended despite various types of errors and failures? The major accomplishments are:

1. Finite-state machine theory and boolean circuit algebra, which relate hardware function to structure.
2. The so-called von Neumann machine, which is the single-instruction sequence stored-program computer.
3. Hardware units for fast arithmetic.
4. Efficient methods of encoding and storing information in various media.
5. Theory of reliable computing systems, including redundant components, reconfiguration, diagnostics, and testing.
6. Methods of synthesizing large complex systems from basic components.
7. Prototypes of multiprocessor machines capable of supporting hundreds or thousands of simultaneously executing processors.
8. Microelectronic circuit technology and computer aided design of very large scale integrated (VLSI) circuits.

PROGRAMMING LANGUAGES AND METHODOLOGY. This area deals with notations for expressing algorithms and data, with efficient translations from high-level languages into machine codes, and with methods of efficiently constructing correct programs. The fundamental questions are: What are the basic data types and operations that arise in various classes of problems and how should they be represented? What are the basic methods of controlling the execution of a computation? How can syntactic descriptions of language be used to construct efficient compilers and optimal code generators? What methods should be used to aid in the process of proving that a program performs its intended function? The major accomplishments are:

1. Procedure-oriented programming languages such as Cobol, Fortran, Algol, Pascal, or Ada. Functional languages such as APL, Lisp, Prolog, and VAL. Object-manipulating languages such as Smalltalk or CLU.
2. Codification of basic concepts of programming languages such as basic data types (e.g., scalars, arrays, records, strings) and control structures (e.g., sequencing, iteration, selection, subroutines, recursion).
3. Theory of compiling and code generation and its application in real compilers.
4. Verification, which deals with establishing that a program's functional specifications are satisfied by its implementation.
5. Syntax-directed editors that monitor program construction and alert the user to potential errors.

ALGORITHMS AND DATA STRUCTURES. This area deals with specific classes of problems and their efficient solutions. The fundamental questions are: For given classes of problems, what are the best algorithms? How much storage and time do they require? What is the tradeoff between space and time? What is the worst case

of the best algorithms? How well do algorithms behave on average? How general are algorithms—i.e., what classes of problems can be dealt with by similar methods? The major accomplishments are:
1. Identification of good and bad algorithms for important classes of problems such as searching, sorting, random-number generation, and textual pattern matching.
2. Identification of general methods applicable across many classes of problems, such as storage of information in tables or lists, graph algorithms, or tree algorithms.
3. Categorizing the effects of data structure on time and space requirements of programs for various classes of problems.

OPERATING SYSTEMS. This area deals with the control mechanisms that allow multiple resources to be efficiently coordinated in the execution of programs. The fundamental questions are: At each timescale in the operation of a computer system, what are the visible objects and permissible operations on them? For each class of resource (objects visible at some level), what is a minimal set of operations that permit their effective use? How can interfaces be organized so that users deal only with abstract versions of resources and not with physical details of hardware? What are effective control strategies for job scheduling, memory management, communications, access to software resources, communication among concurrent tasks, reliability, security, and the like? What are the principles by which systems can be extended in function by repeated application of a small number of construction rules? The major accomplishments are:
1. Prototypes of timesharing systems, interrupt systems, automatic storage allocation, schedulers, and file systems that served as the bases of major commercial systems. Libraries of utilities such as text editors, document formatters, compilers, linkers, and device drivers.
2. Powerful, hierarchical abstraction principles that permit users to operate on idealized versions of resources without concern for physical detail—for example, processes instead of processors, files instead of disks, data streams instead of program input/output.
3. Theories of process management including reliable interprocess synchronization, communication, and deadlock control.
4. Theories of memory management including optimal swapping policies for virtual memory, file access methods, and secondary storage optimization.
5. Hierarchies of directories.
6. Theories of job scheduling, queueing network modeling, and other forms of performance modeling.
7. Models of access control for files owned by specific users.
8. High-level command interfaces that permit users to easily express computations consisting of several components selected from among the files in a system. This includes interactive "windows," command "menus," and pointers such as the "mouse."

NETWORKS. This area deals with the organization of systems comprising interconnected computers. The fundamental questions are: What are the most efficient methods of error checking and correction? of reliably exchanging information across various media (e.g., telephone lines, microwaves, laser optics)? of mediating contention for shared channels? What strategies (protocols) should be used for connecting computers across long distances? short distances? How can the fact that a system is made of components connected by networks be hidden from users who do not wish to see that level of detail? The major accomplishments are:
1. Prototypes for long-haul, computer-to-computer communication networks that served as the bases for commercial networks.
2. Local networks for high-speed connections among close computers, such as Ethernet, pronet, or token-ring nets.
3. Protocols that allow computers to establish and maintain connections across unreliable networks.
4. Protocols that mediate high-speed contention on shared or broadcast channels.
5. Cryptographic protocols that permit secure authentication and secret communication.
6. Structural principles for operating systems that allow hiding the network from those who do not wish to see it.

HUMAN INTERFACE. This area deals with the transfer of information between humans and machines via various human senses and motor skills. The fundamental questions are: What are efficient methods of representing objects and automatically creating pictures for viewing? What are efficient methods for receiving input or presenting output? How can the risk of misperception and subsequent human error be minimized? The major accomplishments are:
1. Core graphics systems for representing objects, for displaying them efficiently, and for creating displays that rotate, translate, pan, and zoom in real time. This includes a wide range of algorithms for constructing pictures from basic components, smoothing, shading, and removing hidden lines.
2. Interactive methods for computer aided design.
3. Advanced forms of input and output such as optical readers, light pens, touch sensitive pads, and the "mouse" pointer.
4. Psychological studies leading to modes of interaction that reduce human error and increase human efficiency.

DATABASE SYSTEMS. This area deals with the organization of large sets of data for efficient queries. The fundamental questions are: What basic models should be used to represent data elements and relations among them? What operations are used to store, locate, retrieve, and match data? How can these operations most efficiently be expressed in language forms? How can high-level descriptions of queries be translated into efficient codes for sifting through the database? What machine architectures lead to the fastest retrievals? How can the data be protected against unauthorized access, disclosure, or destruction? How can large databases be protected from inconsistencies generated by simultaneous access, especially when the data are distributed among many machines? The major accomplishments are:
1. Major models for representing large data sets and relations among the data elements, including the relational, hierarchical, and network models. Special representations of files for fast retrieval, such as inverted trees and associative stores.

2. Design principles for locking records when they can be simultaneously accessed by many users.
3. Design principles for maintaining consistency among multiple copies of data stored on different machines of a network.
4. Principles for preventing unauthorized disclosure or alteration of information in the data base, including protection against statistical inference in real-time query systems.
5. High-performance database machines.

CONCURRENT COMPUTATION. This area deals with the organization of computations that require many processing elements working concurrently. The fundamental questions are: What are the basic models of concurrent computation? What classes or problems are most effectively served by each model? What types of machines are most suited for efficient implementation of programs in each model? What high-level visual tools should be provided so that massively parallel computations can be expressed quickly and correctly? How can the large numbers of resources required in such computations be efficiently managed? The major accomplishments are:
1. General models for parallel computation such as tree machines, mesh array machines, dataflow machines, and communicating sequential processes; new programming languages for these machines.
2. Development of parallel algorithms for important problem classes on these machines; methods of partitioning problems into parts that can be executed concurrently; division of parallel algorithms into time and space complexity classes.
3. Interactive aids for programming and debugging parallel computations.

ARTIFICIAL INTELLIGENCE. This area deals with the simulation of intelligence. The fundamental questions are: What is intelligence? What basic models of intelligence are there and how do we build machines that simulate them? To what extent is intelligence described by rule evaluation and what is the ultimate performance of machines that simulate intelligence by evaluating rules? To what extent is intelligence unpredictable, and can this be modeled by randomness in the machine? The major accomplishments are:
1. Theories of cognition and thought expressed in terms that could be realized by computer.
2. Efficient methods of knowledge representation and searching through knowledge bases.
3. Powerful software systems for logic programming, theorem proving, and rule evaluation.
4. Special applications such as robotics, image processing, vision, and speech recognition.
5. Expert systems based on rule evaluation for simulating expert human behavior in a few narrow domains.

Computer science includes in one discipline its own theory, experimental method, and engineering. This contrasts with most physical sciences, which are separate from the engineering disciplines that apply their findings—as, for example, in chemistry and chemical engineering. I do not think the science and the engineering can be separated within computer science because of the fundamental emphasis on efficiency. But I do believe that the discipline will endure, because the fundamental question underlying all of computer science—what can be automated?—will not soon be answered.

Reference

1. Readers interested in a detailed treatment of the subjects covered here are invited to examine the report of the NSF Computer Science and Engineering Research Study (COSERS), *What Can Be Automated?* ed. B. Arden, 1980, MIT Press.