

# REFLECTIONS ON COMPUTER SCIENCE

*John E. Hopcroft*

Computer Science Department, Cornell University, Ithaca,  
New York 14853

---

## CONTENTS

|                                     |    |
|-------------------------------------|----|
| INTRODUCTION.....                   | 1  |
| THE PAST.....                       | 2  |
| THE FUTURE .....                    | 7  |
| <i>Parallelism</i> .....            | 7  |
| <i>Man-Machine Interfaces</i> ..... | 8  |
| <i>Knowledge Fusion</i> .....       | 9  |
| <i>Applications</i> .....           | 10 |
| CONCLUSIONS.....                    | 11 |

## INTRODUCTION

In 1986, Annual Reviews Inc added the *Annual Review of Computer Science* to its distinguished series of state-of-the-art reviews. This was a significant event for the discipline because the establishment of a new series by Annual Reviews serves as acknowledgment of the maturity and stature of a field.

I have been actively involved in computer science research and education since 1964. Over the years, I have felt the excitement generated by the acquisition of new knowledge in computer science and witnessed the phenomenal growth as the field developed into a vigorous and independent discipline. However, recognition by those outside the field lagged considerably behind growth. I am happy to see that this situation has been rectified. I view the publication of the *Annual Review of Computer Science* as an indication of an increased awareness of the importance of computer science by the scientific community at large. Thus, I am extremely pleased

to have been given the opportunity to write the prefatory chapter for this fourth volume of the new series.

As one of the early contributors to the field of computer science, I can describe the development of computer science into a discipline, and speculate on future areas of growth. In reviewing the history of computer science, one would normally mention such important evolutionary factors as the beginning of artificial intelligence research, Project MAC, Illiac, the role of DARPA, and the influence of major research institutions. However, here I present a slightly different historical view. I omit these well-documented events in favor of other, less well-known developments. We often overlook the fact that what contributes most to the expansion of a discipline is not the major events but the total of all of the advances that take place within it, both major and minor. Here I also concentrate on computer science developments in which I was personally involved.

## THE PAST

The modern era of computing began in 1946 with the development of ENIAC, the Electrical Numerical Integrator and Computer, built at the Moore School of Electrical Engineering at the University of Pennsylvania. ENIAC, the first electronic digital computer, was capable of performing 5000 arithmetic operations per second. Thus ENIAC possessed less computing power than the hand-held calculators used today by high school students; but for its time, ENIAC performed at extraordinary speed and heralded the scientific age of computing. While many researchers and scientists were attracted to computing by exciting computing devices like ENIAC, their energies were channeled primarily toward applying computing instruments to scientific problems rather than toward developing a science of computation.

The science of computation traces its beginnings to the work of Alonzo Church and Alan Turing. It is interesting to note that Turing's paper "On Computable Numbers with an Application to the Entscheidungsproblem," in which he introduced the concept of the Turing machine, was published in 1936, predating ENIAC by ten years. While the roots of computer science go back to 1936, computer science as an academic discipline did not begin until 1962, when the first independent computer science departments were created at Stanford University and Purdue University.

Early computer scientists came to the field with training primarily in electrical engineering or mathematics. Thus it was not surprising that the early discipline placed heavy emphasis on digital switching theory and mathematical logic. Back then, computer science research centered on the design of Boolean switching circuits and the implementation of finite-state

machines. In fact, IEEE's Foundations of Computer Science Conference was initially called the Switching Circuit Theory and Logical Design Conference. Work on switching circuits concentrated on the minimization of the number of components needed to realize Boolean functions in electronic circuits. Work on the implementation of finite-state machines was concerned with the state assignment problem. David Huffman was the key contributor in this area. One of the major intellectual advances of the time was his formulation of the abstraction of the internal state, an idea taken for granted today.

This early work tended to take place in electrical engineering departments; consequently it was viewed as "computer engineering" rather than "computer science." When academic departments of computer science were established, they incorporated the mathematical concept of finite automata but left the circuit realization material to electrical engineering departments. Two books of collected papers were influential in determining the content of early courses in computer science: *Automata Studies*, edited by C. E. Shannon and J. McCarthy, and *Sequential Machines: Selected Papers*, edited by E. F. Moore. Shannon & McCarthy's book contained a paper by Warren McCulloch and Walter Pitts that characterized the behavior of neural activities. Their work led to the regular expression notation, a notation that was subsequently proved equivalent in expressive power to the finite-automaton formalism. Moore's book contained a survey paper on finite automata by Michael Rabin and Dana Scott, which gave a more mathematical treatment to automata theory than found in previous works on finite-state machines. Unlike Huffman's work, which focused on the circuit realization of the machine, their paper focused on the sets of strings that were accepted.

Two other important advances of the time were context-free grammars and Backus-Naur Form (BNF). The context-free grammar was a formal linguistic model developed by Noam Chomsky to capture the syntax of natural language; BNF was a formalism developed by John Backus and Peter Naur to express the syntax of programming languages, such as Algol 60, that were then being developed. The context-free grammar and the BNF formalism were shown to be equivalent. This result, together with the equivalence of regular expression notation and sets accepted by finite automata, placed a coherent structure on automata theory and languages.

Another major influence on academic computer science stemmed from work on Turing machines and computability. In 1964, Juris Hartmanis and Richard Stearns published their paper on "The Computational Complexity of Algorithms" in the *Transactions of the American Mathematical Society*. Although many papers on complexity had been published previously, this paper tied complexity to the atomic steps of the computation

and to the actual number of tape cells used by a Turing machine. Hartmanis & Stearns showed that, with very concrete measures of complexity, modest increases in resources allowed new functions to be computed that could not have been computed using the original resources. This idea sparked enormous interest and led to the study of the intrinsic complexity of problems, an area that would expand significantly over the next 20 years. In fact, it was from this paper that the subdiscipline of computational complexity received its name.

This was the state of the field of computer science in 1964 when I obtained my PhD in Electrical Engineering from Stanford University. Following my graduation, Edward McCluskey at Princeton University hired me to help build a computer science program within their Electrical Engineering Department. My first assignment was to teach a graduate course in theoretical computer science. Since no textbooks were available on this topic, I had to start from scratch. I gathered material on what I considered the most important concepts in the area. In class we discussed the work of McCulloch & Pitts, Backus & Naur, Rabin & Scott, Hartmanis & Stearns, Chomsky, and Turing. Subsequently Jeffrey Ullman and I developed the notes from this course into the text *Formal Languages and their Relation to Automata*, which would shape the way theoretical computer science was taught for the next two decades.

The systems' side of academic computer science lagged behind the theoretical side. Research in computing systems was driven by the need for compilers and operating systems. In the area of compilers, work often focused on the lexical and parsing aspects of compilation. New programming languages were designed; control mechanisms, block structure, parameter passing, and their impact on implementations were explored. Operating systems focused on resource management. It was not until much later, when distributed systems came along with notions of concurrency and synchronization, that operating systems began to flourish as a rich discipline.

Techniques for writing programs and proving them correct began to appear around this time. Much of this work used well-defined mathematical tasks as sample problems—for example, computing the greatest common divisor of two integers, or sorting a set of integers. These tasks possessed a common property: they had very precise specifications. Thus, programming became a science of how to take a formal specification and convert it into a correct program, what today is called programming in the small.

An important next step in the development of computer science was the publication of the first two volumes of the *Art of Computer Programming*, what was to be a seven-volume set by Donald Knuth. Knuth had intended

to write a seven-chapter book on programming, but after the publisher received the first chapter, it was clear that what they had agreed to publish as a book would be more on the order of a small encyclopedia. Fortunately the publisher did not abandon the project. Prior to this time, no organized, comprehensive work had been written on computer algorithms. Knuth's books substantially reduced the entrance cost for researchers in this area and opened up the area of algorithms to scientific inquiry. Thus the domain of computer science was broadened to include the study of algorithms.

During 1970, I spent my sabbatic at Stanford University, sharing an office with Robert Tarjan. Together we worked on various graph algorithms: planarity, triconnectivity, depth-first search, and graph isomorphism. Initially we sought a nonlinear lower bound on the complexity of determining if a graph was triconnected. In the course of our attempts, we discovered linear-time algorithms for testing the planarity and triconnectivity of graphs. Our successes led to the realization that courses in algorithms could be taught on a theoretical basis. *The Design and Analysis of Computer Algorithms* by Aho, Hopcroft, and Ullman was a product of that realization.

In 1970, Stephen Cook at the University of Toronto proved what at first appeared to be an obscure and highly technical theorem; namely, that the problem of whether or not a nondeterministic polynomial time-bounded Turing machine accepts a string could be transformed into the problem of determining if a Boolean formula was satisfiable. This meant that the problem of determining whether a Boolean formula was satisfiable was at least as difficult computationally as any problem that could be solved by a nondeterministic algorithm in time polynomial in the size of the problem description—i.e. any problem in the class that today is known as nondeterministic polynomial time (NP). He presented his results in the paper “On the Complexity of Theorem-Proving Procedures” at the 1971 ACM SIGACT Symposium on Theory of Computing. At first, researchers did not know what to make of this curious result. But a year later, Richard Karp at Berkeley presented his now renowned paper on “Reducibility among Combinatorial Problems” at an IBM symposium at Yorktown Heights. In this paper, Karp showed not only that the problem of satisfiability of Boolean formulas had this property (which today we call NP-completeness), but also that almost all packing, matching, routing, and scheduling problems had it. Thus the notion of classifying problems as complete for a class, such as polynomial time (P) or nondeterministic polynomial time (NP), entered every computer scientist's working vocabulary. Within a year or two, a dozen complexity classes (e.g. PSPACE, polylog time, and exponential time) had been defined, each with its own set of complete problems. Problems were then classified by the complexity class

for which they were complete. Mathematical theories, such as Presburger arithmetic and the theory of reals, were examined to determine their intrinsic complexity, and tools such as relativization were developed.

The exciting work in computational complexity soon attracted many new scientists to computer science, in particular logicians and mathematicians. When these scientists realized the depth of the contributions being made in this new field, they accepted it as a separate and distinct area of inquiry. This was a first step in the process of gaining recognition for computer science as an independent discipline.

During the years that followed, rapid advances were made in a number of areas of computer science, among them randomized algorithms, factorization, cryptography, and polynomial-time algorithms for linear programming. After 1975, an explosion of new subfields developed: relational data bases, semantics, distributed systems, concurrent systems, computational geometry, and VLSI design. By this time, computer scientists could no longer be considered generalists. The field had grown so large that an individual computer scientist could no longer teach every course that needed to be taught. The depth and the variety of the results obtained during the discipline's formative years had led computer scientists to specialize within specific subdisciplines. Computer science had to undergo this process to achieve maturity as a discipline.

Another important step in the growth of computer science occurred during the late 1970s. Jerome Feldman and William Sutherland edited a report to the National Science Foundation entitled "Rejuvenating Experimental Computer Science." The Feldman Report, as it came to be called, documented the need for an infusion of research funds to build up the infrastructure of academic computer science departments. Unlike Stanford, MIT, and CMU, which had been well funded by DARPA, most of the remaining institutions in the country were equipment impoverished; they did not have the resources necessary to carry out broad research programs. To help alleviate this problem, the National Science Foundation responded by creating the Coordinated Experimental Research (CER) Program. This program provided large, five-year block grants to approximately 20 institutions. These funds allowed many computer science departments to acquire equipment necessary to move them beyond the realm of single-investigator projects. Within five years, the CER Program transformed what had been primarily paper-and-pencil research efforts into experimental projects across a wide spectrum of research areas. Many consider the CER Program to have been one of NSF's most successful programs.

By 1985, computer science had firmly established itself and had begun to assert leadership at a national level. No other discipline has grown so

much in so short a period. Now, approximately 25 years after the first computer science departments were created, 125 such computer science departments across the nation produce over 500 PhDs per year. In addition, there are over 40 major research journals devoted entirely to computer science or computer engineering; these journals publish more than 300 issues annually. Computer science has had a major intellectual impact on the academic environment and contributed to the reshaping of such other disciplines as mathematics, linguistics, and cognitive psychology.

## THE FUTURE

Phenomenal progress has been made since the science of computation began approximately 50 years ago. Back then few people could have imagined the state of computing today and the ease with which computers have permeated all aspects of business, science, and education. The same holds true for academic computer science; significant advances have been achieved that have changed the very nature of the discipline. With this in mind, I present some speculations on what the future holds.

I believe that computers will cause an intellectual revolution for mankind. During the next 50–100 years we will witness research advances that will change in a fundamental manner our understanding of knowledge, enhancing our ability to acquire and utilize it. I believe computer science has brought us to the threshold of an information revolution whose consequences will be more profound than the scientific and industrial revolutions combined. But rather than speculate on the distant future, I confine my remarks here to trends in computing that are already visible. In the near term, computer science will be influenced and altered by several forces: parallelism, man-machine interfaces, knowledge fusion, and a trend towards more expansive computer applications. I believe that these factors will have a great impact on research in computer science.

### *Parallelism*

Already numerous parallel architectures have been developed: the single-instruction multiple-data (SIMD) architecture used in the Connection Machine, the message-passing architecture used in the Hypercube, and the shared-memory architectures used in the Butterfly and Encore machines. The driving force behind parallelism is the potential for significant increases in computing power.

Ralph Gomory, senior vice president and chief scientist at IBM, has described computing in terms of marks (those binary quantities that indicate the presence or absence of a signal) and the ease with which marks

are made, read, and erased. He has noted that the phenomenon of doubling computer power every three years resulted from the development of technology that made marks smaller and smaller. In the 1950s a vacuum tube was used to record a single mark; later the transistor was used, then the integrated circuit, and, more recently, very-large-scale integration. Over time, the mark has decreased in size; now it is only a few microns long. Consequently the distance that marks are transmitted has decreased, as has the time required to read or write them. There is no end in sight to the drive to decrease the size of the mark and thereby increase the speed of computation.

However, it is becoming more and more expensive to decrease the size of the mark. As costs increase, it becomes more effective to read and write many marks at once—i.e. to build parallel machines. In the next decade much of our thinking about computation will undoubtedly change from the single-instruction-stream model developed by von Neumann to a parallel model. Reflecting this change, computer scientists will place more and more emphasis on languages and architectures for exploiting parallel computation. Research will focus on the connection between processors and memory, the amount of parallelism that can be extracted from a sequential program, and the linguistic concepts for expressing computations that are carried out on a parallel machine. The switch from the von Neumann model of computation to a parallel paradigm is likely to be rapid. It will take place within the next 10–15 years.

In the long run, massively parallel machines will change fundamentally the way we view computation. For example, problems may have to be reformulated to better utilize parallel technology. Rather than simulating fluid flow by solving differential equations on a highly parallel machine, the problem may be restructured for a cellular-automata model, where each cell will have an internal state and communicate only with neighboring cells. A massively parallel machine may carry out the simulation by assigning one processor to each cell. Such a change in the computational model will produce a significant change in the way scientists view physics. One might be able to prove global properties of material or fluid flow from the local properties of a cell and the interconnection patterns among cells. This is not as farfetched as it may appear; at some level of granularity, the world is discrete, and differential equations are only an approximation to this discrete world.

### *The Man-Machine Interface*

Another area that will play a key role in the future of computer science is computer programming, in essence the man-machine interface. While initially computers were programmed using machine code, the first major

advance over machine code was assembly language. Although it involved almost a one-to-one translation of machine code, assembly language's ability to use mnemonics for operations made it a major advance. In addition, programmers were no longer required to allocate physical locations for instructions and storage in memory; instead they worked within the space of logical addresses and left the task of making physical assignments to the computer. The next important advance in programming was the development of high-level languages, such as FORTRAN. High-level languages made it possible to write arithmetic expressions in a familiar notation. This, along with subsequent advances in control structures, block structures, data abstractions, etc, tremendously improved programming productivity. The sequence of languages that followed—Algol, Pascal, etc—exhibited minor advances compared to the fundamental advance of FORTRAN.

Programming is a challenging intellectual task. As such it is a major barrier to the full use of computers in everyday life. To overcome this barrier, it may be worthwhile to learn from the industrial revolution. The industrial revolution produced fundamental changes in production methodologies, and it had a profound effect on quality of life and on technology. By breaking up the work done by skilled artisans into simple, well-defined tasks, laborers with minimal technical skill could produce the same high-quality products as artisans. The skill level for computer programming today appears to be moving in the opposite direction, requiring higher and higher skill levels. Unless we reverse this trend and make computers accessible to the average worker, many of our advances will be of little benefit to society as a whole. Procedures for accessing computers are too complex for the average citizen. Current man-machine interfaces must be altered to allow those with less skill to interact easily with computers, at all levels. We have seen the beginning of this kind of process with systems like the Macintosh computer and with spreadsheets. It is likely that spreadsheet technology has expanded the use of computers more than any other invention; it may provide us with the key to make computer systems accessible to large numbers of people. Just as during the industrial revolution, this increased accessibility will open up new and important areas for growth. Thus the computer revolution is only now beginning.

### *Knowledge Fusion*

A deeper understanding of the process of knowledge fusion will be achieved in the near future, and this will have a significant impact on computer science. Humans instantaneously comprehend images; when we see a tree, we immediately recognize it as one. Since trees come in an infinite variety of

shapes, it is obvious that our recognition does not depend upon geometrical pattern-matching. Moreover, we recognize trees we have never seen before. What makes a tree a tree is not its shape and texture alone, but something more intrinsic. Our recognition of the tree involves the fusion of many bits of information, some that support the decision for treeness and some that do not. By some as yet unknown technique, humans immediately combine various sensory data to arrive at a correct decision. This combination algorithm must be far from the bottom-up methods of object recognition used in computer vision systems today, which identify features such as edges and corners and try to combine these features into more complex patterns. Somehow the human computer possesses top-down information about what is most likely in view. (For example, if one is outdoors, an object is more likely to be a tree than if one is indoors.) The human computer simultaneously processes bottom-up information about what it is seeing and top-down information about what it expects to see; then it fuses these two computations. The fusion probably occurs at many different levels rather than at a single interface. Our recognition processes are rarely fooled; and when they are, the error is usually only momentary. For example, when looking out a train window, we may believe that our train is moving when a train on an adjacent track starts to move. However, since additional information does not jibe with this belief our brain immediately corrects its impression. No computer program yet written has capabilities so advanced, but I foresee developments like this in the near future.

### *Applications*

Computer usage was at first limited to scientific computations and the management of large, similarly structured data; for example, computers were used to calculate trajectories and track payroll files, insurance records, and airline reservations. Today there are applications on the horizon whose sophistication far exceeds these early applications. A single earth-scanning satellite may transmit 16,000 photographs a day, each covering a  $150 \times 150$  square mile area; merely cataloging this data requires considerable computer automation. Imagine a computer program that will scan these images and report the occurrence of anything atypical, such as a major earthquake or a reactor meltdown.

An application that promises to improve manufacturing productivity dramatically is electronic prototyping, which allows us to substitute computer models for physical ones. Using electronic prototyping, one can design an object such as a multifingered gripper and test its design by programming the gripper to manipulate an object or perform an assembly. Since the gripper is only a computer file, it can be edited easily to change its size, shape, or finger configuration. This ability to explore a wide design

space before building an object will have a tremendous impact on the variety and quality of products. Standardized design descriptions will lead to highly automated production with greatly reduced costs.

The concept of electronic prototyping illustrates the importance of the information content of a physical object or a design. It is in the management and transformation of information that computers will have their greatest impact. For example writing, a tool for storing and communicating ideas, has been by and large a linear medium; but we are now seeing other forms of writing, like HYPERTEXT, where the order in which we wish to access information can be restructured. The computer gives us the opportunity to interact actively with written work, including pictures and 3-dimensional models. The computer allows us to interact with what might be called artificial realities. The ability to represent knowledge in procedural form may become as important as the ability to record knowledge in written form.

The pervasiveness of computer applications is bound to have a profound impact on computer science, expanding future research beyond the narrow confines we know today.

## CONCLUSIONS

Computer scientists can look back with pride on the last 50 years, the years during which computer science evolved from an interesting area of research into a mature and independent discipline. Many changes took place during this time, many research avenues were explored, and much knowledge was gained. It was an exciting period, but the next 50 years will be even more exciting for computer science.

The course that computer science will follow in the near future seems clear. Parallelism will be a driving force for change in the field; advances in high-level languages will reduce the complexity of programs and allow more people to interact with computers; computer capability will increase substantially as advances in knowledge fusion occur; and computer applications will continue to expand across a broad spectrum of areas.

My long-term vision for computer science is not as clear. I believe that revolutionary advances in computing principles lie just out of sight. During the next century, we will begin to unravel these mysteries.

In the 1600s Newton revolutionized our understanding of the physical world by formulating the most elementary laws of motion. Today, less than 350 years later, we have developed a deep understanding of the nature of matter, quantum mechanics, and the creation of the universe. Computing is on the brink of a similarly Newtonian development. Fundamental computational principles lie on the horizon that are capable of

vastly changing the way we process information. I predict exciting times ahead for computer science.

### *Literature Cited*

- Aho, A. V., Hopcroft, J. E., Ullman, J. D. 1974. *The Design and Analysis of Computer Algorithms*. Reading: Addison-Wesley. 470 pp.
- Cook, S. A. 1971. The complexity of theorem proving procedures. *Proc. 3rd ACM Symp. on Theory of Computing*, pp. 151–58
- Feldman, J. A., Sutherland, W. R. 1979. Rejuvenating experimental computer science. *Commun. ACM* 22: 497–502
- Hartmanis, J., Stearns, R. E. 1965. On the computational complexity of algorithms. *Trans. AMS* 117: 285–306
- Hopcroft, J. E., Ullman, J. D. 1969. *Formal Languages and their Relation to Automata*. Reading: Addison-Wesley. 242 pp.
- Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, ed. R. E. Miller, J. W. Thatcher, pp. 85–104. New York: Plenum
- Knuth, D. E. 1973. *The Art of Computer Programming*, Vol. 1. Reading: Addison-Wesley. 634 pp. 2nd ed.
- McCulloch, W. S., Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* 5: 115–33
- Moore, E. F., ed. 1964. *Sequential Machines: Selected Papers*. Reading: Addison-Wesley. 266 pp.
- Rabin, M. O., Scott, D. 1959. Finite automata and their decision problems. *IBM J. Res. Devel.* 3(2): 114–25
- Shannon, C. E., McCarthy, J., eds. 1956. *Automata Studies*. Princeton: Princeton Univ. Press. 285 pp.
- Turing, A. M. 1936–1937. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* 42: 230–65