COMPUTER SCIENCE: THE EMERGENCE OF A DISCIPLINE

The continued rapid development of computer science will require an expansion of the science base and an influx of talented new researchers. Computers have already altered the way we think and live; now they will begin to elevate our knowledge of the world.

JOHN E. HOPCROFT

It is a great honor to be a recipient of the 1986 Turing Award. Along with the recognition provided by the award comes the opportunity to present this lecture, and so to speak not only to computer scientists, but also to policy makers and to other members of the scientific establishment. I would like to start by relating some of my experiences in the field of computer science, and then to make some recommendations about the future development of the discipline.

I began my professional career in computer science in 1964, when computer science was just beginning to establish itself as an academic discipline. Having been a part of the academic community during the formative years of computer science, I have been in a fortunate position to observe it as it evolved and to watch it as it matured and developed. I have a great fondness for the field, and I would like to see it continue to flourish and grow. Even though computer science has emerged as a mature discipline, strong leadership and direction within the profession are still of great importance in order for it to contribute fully to science and society.

What has impressed me most during my years in computer science is the level of commitment of the participants. In the early years, I saw strong individual commitment. Later, I saw institutions joining forces with individuals to form strong support systems for computer science. Today, technology is advancing the discipline so rapidly that the combined commitment of individuals and institutions is no longer adequate to meet the challenges created by the expansion of knowledge. The demands of industrial research laboratories and academic institutions far surpass the current resources for producing the needed pool of talented researchers. To reap the maximum benefit from the scientific and technological advances of computing, a national commitment must be made and sustained.

Before expanding on this need, let me first relate some of the events in my career that have brought me to this position. When I received my Ph.D. in electrical engineering from Stanford University, my education had included only one course in computing, which was taught by David Huffman. My last year at Stanford coincided with the year Huffman spent there, and it was from him that I received a basic introduction to switching circuits, logic design, and the theory of computation. During the spring of that same year, Edward McCluskey was recruiting faculty for his Digital Systems Laboratory at Princeton. By chance, he happened to be telephoning Bernard Widrow at Stanford to discuss prospective Ph.D. candidates just as I was walking by Widrow's door. When Widrow saw me, he motioned me into his office and handed me the telephone, and we arranged that I would visit Princeton. The fact that I had no formal education in computer science, except for Huffman's course, did not deter McCluskey. At that time, few people had an educational background in computing. After my visit, I was sufficiently impressed by McCluskey's commitment to computer science and with the opportunity he pre-

© 1987 ACM 0001-0782/87/0300-0198 75¢

sented me to begin a career in the developing science of computing that I accepted his job offer as soon as it was extended. This decision significantly altered my career plans, for prior to McCluskey's telephone call, I had planned to teach electrical engineering on the West Coast.

My arrival at Princeton in the fall of 1964 occurred at a time when a dramatic change was taking place in the computing field. Much of the course content in computer science had focused on the design of circuits for digital computers and minimizing the number of transistors needed to build these circuits. By the mid sixties, however, technology had advanced to the point where transistors were about to be replaced by computer chips with as many as a hundred components per chip. Thus, minimizing the number of transistors was no longer relevant. As you can imagine, this had profound ramifications for what was then called computer science; existing courses were about to become obsolete, and new ones had to be developed.

Princeton asked me to develop a course in automata theory to expand the scope of the curriculum beyond the digital circuit design course then being offered. Since there were no courses or books on the subject, I asked McCluskey to recommend some materials for a course on automata theory. He was not sure himself, but he gave me a list of six papers and told me that the material would probably give students a good background in automata theory. McCluskey's list included works by Warren McCulloch and Walter Pitts, John Backus and Peter Naur, Noam Chomsky, Michael Rabin and Dana Scott, Juris Hartmanis and Richard Stearns, and, of course, Alan Turing.

At the time, I thought it strange that individuals were prepared to introduce courses into the curriculum without clearly understanding their content. In retrospect, I realize that people who believe in the future of a subject and who sense its importance will invest in the subject long before they can delineate its boundaries.

When I look back on the material in that early course in automata theory, I am struck by the diversity of these sources. In 1943 McCulloch and Pitts, working in neurophysiology, published a paper on a logical calculus for describing events in neuron nets. These events were series of electrical pulses and could be viewed as strings of zeros and ones. The paper had a notation for describing how these strings of zeros and ones combine in neurons to produce new strings of zeros and ones. This notation was subsequently developed into the language of regular expressions for describing sets of strings.

Rabin and Scott were mathematicians who developed a model of a computer with a finite amount of memory. They called this model the finite-state automaton, and showed that the possible behaviors of finite-state automata were precisely those behaviors that could be described by the regular expressions that grew out of the work of McCulloch and Pitts. This confluence of ideas from two widely different disciplines helped convince early computer scientists of the importance of regular expressions and finite automata.

Chomsky, a linguist, had been studying the syntax of natural languages. In the course of his work, he developed the concept of a context-free grammar. At about the same time, two computer scientists, Backus and Naur, were attempting to develop formalisms for describing programming languages. Before 1960 programming languages were defined by lengthy and often incomplete verbal descriptions. Inconsistencies in various implementations of a language often made it difficult to change software between systems. Backus and Naur developed a formal notation for describing the syntax of various programming languages. Amazingly enough, their notation was equivalent to the context-free grammars developed by Chomsky.

Turing had in 1936 introduced a simple model of a computing device, which is now known as the Turing machine. This device was simple enough that there could be no question that any function computed by it was computable. However, Turing argued further that his model could compute every function considered computable. Simply put, any computational process that could be carried out could be programmed on the Turing machine. Today this hypothesis is universally accepted, and the Turing machine is the foundation of modern computability theory.

Turing's work might have remained in the realm of mathematics and logic were it not for a seminal paper on the computational complexity of algorithms by mathematicians Hartmanis and Stearns. They measured the complexity of an algorithm by the number of steps needed for its execution and used this method to develop a theory of complexity classes. This paper sparked the imagination of many computer scientists and led to the establishment of complexity theory as an integral part of the discipline.

Certain research papers are important not only for their technical contributions, but, more importantly, because they provide a conceptual view or establish a paradigm for research. The work of Hartmanis and Stearns attracted researchers and focused attention

on the topic of complexity. Among the more significant advances that resulted were the classification of the complexity of most major mathematical theories, the reducibility of many combinatorial problems, the concept of NP-completeness, and a deeper understanding of concepts such as randomness.

And so the early automata theory course served to emphasize two important aspects of computer science: the way it has used a multiplicity of ideas from diverse fields to develop and expand, and the way basic research can compound and escalate advances in computing. On a more personal note, the course had a tremendous impact on my career. Through it I met Jeffrey Ullman and Alfred Aho, with whom I subsequently collaborated for many years. *Formal Languages and Automata Theory*, which I wrote with Ullman, also evolved from this course.

In the spring of 1967, Princeton asked me to run a seminar series. As is often the case, funds for the seminars were limited. The intention was to invite local speakers; however, there was just enough money to sponsor two outside speakers from within a 200-mile radius. Drawing from my interest in automata theory, I invited Chomsky and Hartmanis, both of whom agreed to speak. Hartmanis's visit had a significant impact on my career. During the course of his visit, he asked me about prospective Ph.D. candidates for faculty positions at Cornell. Our ensuing discussion that evening on the importance of the science base for computing and the fact that Cornell had established an independent computer science department led me to ask if I might be considered for one of the positions, instead of a new Ph.D. candidate. After visiting Cornell, I immediately accepted their job offer.

I had first learned of Cornell's commitment to computer science from a news article that had appeared the year before Hartmanis's visit—1966. At that time computer science departments were rapidly being established at a rate that exceeded the supply of faculty to staff them. The article discussed Cornell's recognition of this problem and their efforts to solve it. Three faculty members from the mathematics and engineering departments had persuaded the Sloan Foundation to donate one million dollars to develop an independent computer science department to produce the Ph.D.'s needed to staff the computer science departments being created at other institutions.

By the time I arrived at Cornell in 1967, automata theory and complexity theory were established as parts of the computer science curriculum. And so shortly after switching institutions, I decided to switch fields and began to work on algorithms and data structures. I believed that the methodology of theoretical computer science could be used to develop a science base for algorithmic design that would be useful to the practitioner.

During the 1960s, research on algorithms had been very unsatisfying. A researcher would publish an algorithm in a journal along with execution times for a small set of sample problems, and then several years later, a second researcher would give an improved algorithm along with execution times for the same set of sample problems. The new algorithm would invariably be faster, since in the intervening years, both computer performance and programming languages had improved. The fact that the algorithms were run on different computers and programmed in different languages made me uncomfortable with the comparison. It was difficult to factor out both the effects of increased computer performance and the programming skills of the implementors—to discover the effects due to the new algorithm as opposed to its implementation. Furthermore, it was possible that the second researcher had inadvertently tuned his or her algorithm to the sample problems. Conceivably, if the two algorithms were run again on another sample set of problems, the original algorithm might outperform the newer one.

I set out to demonstrate that a theory of algorithm design based on worst-case asymptotic performance could be a valuable aid to the practitioner. First, a size was associated with each instance of a problem; then the complexity of an algorithm was measured by calculating the rate of growth in computing time as a function of the problem size. Owing to problems in estimating input distribution, the worst-case analysis over all inputs of a given size was adopted as the measure of complexity.

There were a number of problems associated with worst-case asymptotic complexity. Some of the asymptotically optimal algorithms were numerically unstable; others were sufficiently complex that they were unlikely to be programmed. Further, the expected time for realistic input distributions would be a more reasonable measure of complexity. But the paradigm of worst-case asymptotic complexity provided a mathematical criterion for measuring the goodness of an algorithm, making it possible to ask such questions as "What is the optimal algorithm for a problem?" and "What is the intrinsic complexity of a problem?" The idea met with much resistance. People argued that faster computers would remove the need for asymptotic efficiency. Just the opposite is true, however, since as computers become faster, the size of the attempted problems becomes larger,

thereby making asymptotic efficiency even more important.

In early 1970 I took a year-long sabbatical at Stanford University, where I met and shared an office with Robert Tarjan, a second-year graduate student. The research recognized by the 1986 Turing Award took place during that period of collaboration. We worked together on a number of graph algorithms, including graph connectivity and planarity testing. We adopted the philosophy of designing for worst-case performance and demonstrated that a few simple techniques could be used to construct efficient algorithms in many areas of computer applications. We were able to prove that some of our algorithms had worst-case performance optimal within a constant factor, and also that the performance of the resulting algorithms far surpassed that of existing algorithms. Our planarity algorithm was able to test graphs with 1000 vertices in about 10 seconds—two orders of magnitude faster than existing algorithms. Our results attracted many researchers whose efforts created new data structures and design techniques. Today this area is an integral part of computer science. Again, it was not one specific algorithm that was of fundamental importance, but rather that our work captured the imagination of others. It also attracted the attention of bright young researchers who went on to establish data structures and algorithms as an important subdiscipline.

I would like to make some observations and recommendations for the future of computer science. During my career, I have seen the commitment of individuals and institutions grow and expand as computer science has grown and expanded. I believe, however, that much of the credit for the emergence of computer science as a discipline rests with the dedication and commitment of a relatively small number of researchers who had a vision of the potential of computing and the perseverance to make this vision a reality. It is now our responsibility to formulate a new vision, to shape the goals for the next generation of researchers. It is important that computer scientists share this new vision and make it a reality.

Today, computers have penetrated almost every aspect of modern life. They are used in agriculture, communication, education, manufacturing, and medicine. They are used to predict weather, to optimize food production, to control satellites in space, to develop new drugs, and to manufacture fuel-efficient automobiles. In medicine, their use in tomography allows precise imaging of vital organs to aid diagnosis and treatment. In the business world, they are used to route messages, handle commercial transactions, and access databases. They are advancing the frontiers of knowledge in physics, chemistry, and biology. Computers will also play a vital role in the economic revitalization of this country.

Computers have already made a major impact on the way we think and live. However, I envision an even greater impact. The potential of computer science, if fully explored and developed, will take us to a higher plane of knowledge about the world. Computer science will assist us in gaining a greater understanding of intellectual processes. It will enhance our knowledge of the learning process, the thinking process, and the reasoning process. Computer science will provide models and conceptual tools for the cognitive sciences. Just as the physical sciences have dominated man's intellectual endeavors during this century as researchers explored the nature of matter and the beginning of the universe, today we are beginning the exploration of the intellectual universe of ideas, knowledge structures, and language. I foresee significant advances continuing to be made that will greatly alter our lives. The work Tarjan and I started in the 1970s has led to an understanding of data structures and how to manipulate them. Looking ahead, I can foresee an understanding of how to organize and manipulate knowledge.

In the sixties, a change in technology broadened the scope of computer science from circuit design to computation. The tremendous increases in computing power that are on the horizon today will once again fundamentally change computer science. No one can foresee the precise details, but we can sense the potential for understanding the semantics of language, abstraction, and knowledge representation. Just as early researchers were willing to invest their energies in computer science before fully understanding the details, we must also commit ourselves to the future of computer science before fully discerning its shape.

Today, there are signs that computer science is turning to applications areas. As it contributes its models, tools, and techniques to these new fields, they in turn will contribute new ideas and methodologies that will greatly enrich and expand the scope of computer science. Potentially, we are at the threshold of a new era of growth of the science. However, two areas impede our progress over this threshold.

First there is the inadequate size of the science base, as well as the lack of sufficient researchers to expand it. Despite a sizable body of knowledge, tools, and techniques, the science base of computer science is not developing as rapidly as it should. With computer technology advancing so quickly, it

is easy to lose sight of the importance of developing the underlying science base for computing. There is too great a temptation to focus simply on writing software. As we build larger and more complex systems, we must develop the conceptual tools that will allow us to comprehend the essence of a task and to develop the software tools needed to create complete systems. We have not been able to utilize fully the benefits of computing precisely because we have failed to develop the science base necessary for constructing reliable and user-friendly software systems quickly and economically. Existing programming languages are not satisfactory for the task; they appear to be requiring higher and higher skill levels. We must find ways of communicating with computers that lower the required skill level, just as the assembly line lowered the skill level needed in the production process. A factor of 100 improvement in software productivity would allow us to design, implement, and install a given system in two or three days rather than over the course of a year. Such improvements depend on the development of a science base for programming environments, software development, and man–machine interfaces.

Clearly, the science base established by early researchers has been beneficial. Each of the six papers covered in that early automata theory course added considerably to it. For example, extensions of the work of Backus and Naur on formal descriptions of programming languages allow us to automate the construction of the parsing component of compilation. An early Fortran compiler took 20–50 man-years of effort. Today we assign undergraduate term projects that involve constructing a compiler for a language that is conceptually far more sophisticated than Fortran and see it completed within the term. This dramatic improvement is attributable to the development of a science base in formal languages. This same science base allows us to construct compiler compilers and to tailor languages to specific needs. It allows a chemist to write in a language where the data items are molecules and valences rather than integers and strings. Similar contributions have been made in databases, concurrency, algorithms, VLSI design tools, and many other areas. But greater efforts must be made.

Why is the science base lagging behind the technological base? The field of computer science has grown explosively, more rapidly than any other discipline in history. It is unique in that it evolved from researchers from diverse backgrounds instead of emerging from an existing discipline. Other fields, such as molecular biology, had the advantage of emerging from broader disciplines that could contribute researchers of all ages, along with resources and structures. Computer scientists came from many backgrounds and have not been able to bring the support structures of a mother discipline with them. Consequently, computer science began without a sufficient number of recognized senior scientists and without existing support structures to facilitate and channel its growth.

The demand for computer science researchers is ever increasing. Even within the profession, there is competition for researchers. The demands of educational institutions and research laboratories for computer scientists are growing faster than the field can build the infrastructure necessary for producing the needed pool of talent. The shortfall in computer science faculty, in particular, has serious ramifications. Unless an investment is made to provide sufficient researchers in our educational institutions, we will fall even further behind in trying to meet this ever-growing demand for computer scientists.

Universities that realized the importance of computer science early on and acted were able to establish departments of the highest quality. Generally, it has been difficult for institutions that started later to catch up. A similar situation is arising on the national scene. Today, there is a global struggle for technological and economic leadership. Computing will play a key role in this struggle. Unless we develop a national policy to support computer science, we will, by our own inaction, allow other countries to establish leads in computing that cannot be overcome.

A national commitment to computer science must be made and sustained. We must develop innovative programs to provide an adequate talent pool of researchers, use existing researchers more effectively, and develop environments to foster research in all aspects of computing. We must increase public awareness of computer science. And we must convince policymakers of the importance of a full-scale commitment to computer science.

CR Categories and Subject Descriptors: A.0 [**General Literature**]: General—*biographies/autobiographies*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.2 [**Discrete Mathematics**]: Graph Theory; K.2 [**Computing Milieux**]: History of Computing—*people*

General Terms: Algorithms, Design, Performance, Theory

Additional Key Words and Phrases: John E. Hopcroft, Robert E. Tarjan, Turing Award

Author's Present Address: John E. Hopcroft, Dept. of Computer Science, Cornell University, Ithaca, NY 14853.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.