What Can Be Automated?

The Computer Science and Engineering Research Study

(COSERS)

Bruce W. Arden, Editor

© 1980

The MIT Press
Cambridge, Massachusetts, and London, England

Joseph F. Traub
Columbia University

Stephen S. Yau
Northwestern University

What Can Be Automated?

The *CO*mputer *S*cience and *E*ngineering *R*esearch *S*tudy

(COSERS)

It is truly difficult to capture with a single question the essence of research in a diverse and very active area of science and technology, but the query in the title comes very close. This questions was first posed by the late Professor George Forsythe of Stanford University as the underlying rationale for research in computer science, and this study supports his perception. The many different questions, which are posed explicitly and implicitly in these chapters, can be interpreted as special cases of "What Can Be Automated?" The growing realization that the answers to this question are of increasing importance in a crowded world with many interdependent segments is one of the main motivations for this effort.

The initial planning for the Computer Science and Engineering Research Study (COSERS) commenced in 1974 and the study was subsequently undertaken in 1975 with the goals of completing the work in three years and publishing the report in 1978. The National Science Foundation (Grant GJ-43540 and DCR74-18460) has supported a major part of this effort in the belief that a readily accessible record of past research in computer science and computer engineering, as well as some projections of future research directions, would be useful. These research areas are still relatively new and not well defined in the minds of many technical people not actively involved in such research.

In language often used in the implementation of large computer projects, the schedule "slipped." Self-description is an important aspect of several areas of computing research, and it may be appropriate that the generation of this report should exhibit some of the same problems that the described research addresses.

Needless to say, the subject areas considered are rapidly changing. Research in computing is truly a moving target. New chapters, additions and major revisions appeared well into 1979.

At the outset, the decision was made to heavily edit the submitted manuscripts in an effort to remove jargon, to introduce some uniformity of style, and to avoid "journalese" that would tend to restrict readers to those with some familiarity with the subjects. In short, the goal was to produce prose that would be readily accessible to a layman having a general knowledge of science and technology. Such an excursion into language transformation and semantics is an ambitious and humbling experience, it renews one's appreciation for the power of succinct notation. The expansion of technical terms into conventional prose inevitably loses precision the author intended even though, up to a point, it improves comprehensibility for a technical layman. There is an elusive, optimal line between lost precision and increased general comprehension which was crossed and recrossed many times - testing the patience of all concerned. It is hoped that the result of this trying process has, to a large extent, achieved the original goals.

Some time after the generation of chapters had started, many of the contributors advocated that the drafts be maintained by computer text-processing and that the published version be phototypeset under computer control. Not only would this facilitate the editing and revision but it would illustrate an important application of computers. In retrospect this was a very good decision but there were difficulties along the way with changing text processing programs and the unfamiliarity of conventional editors and publishers with the media and programs used in computer phototypesetting. It is a powerful, complex tool, but it requires an understanding of computers and programming, if not expertise. Here again, the study illustrated problems shared with many other computer applications.

The Steering Committee, whose names appear on the title page, represent university, industrial, and governmental research activities but, perhaps more relevant, they share a belief in the importance of research in the various aspects of computing. Their support of such research is apparent from their many personal contributions and their willingness to devote some time to COSERS is further evidence of their dedication. The committee suggested study personnel and an organization for the effort; many of the members actively served in liaison roles with specific panels.

There are eighty individual contributors to this report. These authors are grouped into subject area panels which are listed at the beginning of the chapter for which they were responsible. The chairmen of these panels were responsible for the selection of contributors and for the organization of the resultant chapter. Much of the burden of producing this publication has fallen to these chairmen; their commitment to obtaining an accessible exposition of research in their subject areas has sustained the study.

In addition to the contributing authors there have been many readers whose criticisms of specific sections or points have been helpful. On behalf of the panels this assistance is gratefully acknowledged. In a few instances the interaction of readers was such that it was tantamount to participation in the panel and these readers are listed with the panel contributors.

With the exception of the Overview, Applications, and COSERS Statistics, each chapter attempts to cover the significant research results in the area and to identify outstanding problems or, in other words, future directions. There are inevitably major differences in the chapter formats but, in all cases, the references are representative, not comprehensive. For the most part, references to a chapter-end bibliography are simply indicated by numbers in the text although some results are so closely associated with particular investigators that identification by name is expedient.

As mentioned, the study was made possible by the support of the National Science Foundation. In particular, Dr. Fred Weingarten has been of great assistance. Also Bell Laboratories has assisted in several important ways. The enthusiastic support of Steering Committee member Dr. William O. Baker has been an inspiration to everyone with whom he interacted. The text processing and phototypesetting was done at Princeton and Bell Laboratories on computer systems operated under UNIX^{TM} a widely used operating system developed by the Laboratories. The expertise of Dr. Michael Lesk, one of the designers of the UNIX text processing software, was an invaluable asset in the preparation of camera-ready copy for publication.

Several people at Princeton have been involved in editing, text processing and figure preparation. Mr. Roy Grisham initially edited much of the material and entered it in computer files. Mrs. Betty Steward, the departmental secretary, has handled the correspondence and the many organizational details associated with such a large, protracted effort.

In summary, it has been a lengthy study of rapidly changing subjects. Only the commitment and the enthusiasm of investigators and the promise of increasingly beneficial computer applications seem constant. Computing research and development have made systems of enormous complexity possible, but many questions need to be answered before the exploitation of the full potential is commonplace.

B. Arden
Princeton University
July, 1979

1 COSERS OVERVIEW

Bruce W. Arden
Princeton University

COSERS Overview

It would be inappropriate to consider at length the impact digital computers have had on the industrialized world. Many books, special journal issues, and popular articles have been devoted to the "computer revolution" or the "age of the computer."[3,4] According to a recent prediction, the U.S. gross national product attributable to computers will exceed that of automobiles within a decade. The growing interdependence of the various segments of society increases the need for the transmission and processing of information. This trend, which is relatively energy conservative in comparison to means of coordination which depend on the transportation of people, will very likely lead to the realization of the prediction.

The explosive growth of computer-related business attests to the pervasive applicability of computers. The expanding market for innovative computer programs, technology, and service has provided the incentive for corporations to actively engage in computer oriented research and development. Indeed, it is occasionally argued that this market-motivated development makes it less necessary to support research in computer science and computer engineering from nonindustrial sources. The acceptance or rejection of this argument depends upon an assessment of the importance to society of sponsoring long term research. For the most part, industrial research personnel concentrate on studies motivated by the prospect of imminent commercial exploitation. Is it important that these efforts be augmented by parallel, but not directly applicable, work in, say, the development of an underlying theory, the design of radically different computer organizations, or the design of specialized programming systems? A standard answer to this question is that the development and augmentation of a body of knowledge, in itself, is sufficient justification for such a concurrent effort. This answer is well supported by the history of science, which describes many instances where undirected research has radically altered scientific thought. For example, in this century studies of spatial and temporal relativity, physical measureability, and mathematical provability have markedly changed scientific research. One of the exciting prospects of the computer era is that equally influential concepts relating to feasibility may be formulated.

In an ideal world such goals would be sufficient to justify the support of research, but in a pragmatic world, beset by many problems, the case is more persuasive if the avoidance or solution of societal problems can be anticipated as a result of the work. Such results are almost as hard to predict as conceptual breakthroughs, but one thing is certain: decisions based on computers and communication will shape the society of the future. With many people looking beyond near term developments, the chances of undesireable side effects of the computer age will be reduced.

Looking back to the commencement of an earlier "age," automobiles have had remarkable applicability; they have had a profound effect on the social system of the present automotive era. Automobiles have been developed and refined along certain lines in response to direct, marketable applications. One wonders whether, in the early years of this development, a vigorous research program involving effects and alternatives to these technological developments could have shaped a better transportation system, one less plagued by unanticipated problems. While the analogy is imperfect, the computer revolution does have the potential to shape future society in ways that will probably be as profound as those of the automobile. In anticipating such an impact, it seems all the more important to perform the long-term study and research, in order to identify and avoid, if possible, the negative social side effects that have accompanied other technological revolutions.

In 1966 C. P. Snow[11] looked further back in history and made a prediction that could be made with even more certainty today:

> [The computer revolution] is going to be the biggest technological revolution men have ever known, far more intimately affecting men's daily lives, and, of course, far quicker, than either the agricultural transformation in Neolithic times or the early industrial revolution which made the present shape of the United States.

In the subsequent discussion of the impending revolution, he issued a warning:

> We shall also need many people of different abilities, who are at every step of the way studying, controlling, and humanizing [the computer revolution's] effects. There was some excuse for our ancestors' not foreseeing the effects of the first industrial revolution. There is no excuse this time.

The primary purpose of this report is to describe, for technical laymen, what has been accomplished under the headings of "computer science research" and "computer engineering research" and to make some limited predictions of the future. This introduction, then, is intended to cover the underlying definitional issues, to describe the trends that are relevant to the research, to review the various chapters of this report, and to relate them to each other. In any major group-writing effort, it is necessary to divide the overall subject area, to reduce it to manageable subtasks. Here, too, this has been done at the risk of creating boundaries that are somewhat arbitrarily drawn. Many researchers feel strongly that this partitioning is especially inimical to the study of computers and computing. They believe that much of the constructive work that has been done is due to the specialty-bridging nature of research in computer science and computer engineering. The intellectual excitement that arises from research challenges and achievements in these rapidly changing subject areas is hard to convey. It is difficult to find a popular level of exposition, as one often can in the physical sciences. Nonetheless, these chapters do reveal the enthusiasm of the many contributing authors and, importantly, the developing cohesion of the topics.

About Names and Labels

The Latin verb *computare* has the meaning of arithmetic accounting or, to use another old word, of *reckoning*. If the meaning had persisted in its various derivatives, "numerical computation" would be redundant. But certainly a major, if not dominant, aspect of computing today is nonnumerical. This early inclusion of non-arithmetic manipulation of symbols as an attribute of *computation* is an important instance of the rapidly expanding meanings of the derivatives of the Latin word.

The terms *computer science* and *computer engineering* are now widely used to categorize the work described in these chapters. Both are well-established, and usage has given them connotations beyond that suggested by the word definitions themselves. Along these lines, the adjective *computer* suggests a machine—or, to use the current jargon, "hardware." For this reason, many people in the field think the more general adjective, *computing*, is more appropriate. This term can include machines as well as other computer-related topics such as algorithms and programs. An even more general candidate is *information science*. But in this country, that term has also been used for the study of libraries and communication media, as well as such specific topics as the analysis and processing of continuous-time electrical signals. It is often argued that the subject *is* that comprehensive, and the word *computer* (or *computing*) is too closely identified with numerical computation. The fact that computers are general symbol-processing devices, with the usual arithmetic computation as an important special case, is more apparent when the word *information* is used. However, the difficulty with such broadly generic expressions is that they do not make useful distinctions. Interestingly, a related, newly coined word, *informatics*, is gaining general acceptance in Europe. In a way, the use of a new word as a label for a new subject area is desirable. The semantic content that such a word or phrase acquires is not prejudiced by the common meaning of the words before they have assumed an area-labeling role. In the United States, however, the adjective *computer* preceding "science" or "engineering" is now well established.

The argument over labels raises again the old debate about the distinction between "science" and "engineering." Science is commonly defined as a systematic body of knowledge with accompanying explanatory models or theories. In scientific research the emphasis is on the augmentation, by discovery, of this knowledge and the concomitant extension or revision of the descriptive theories. It is usually understood that the observations comprising the body of a specific science are obtained from given, natural systems which are not subject to human modification. That is, the facts are observations of given, natural phenomena. Parenthetically, it is worth noting that the explanatory theories vary greatly in extent, formality, and style from one science to another. At one extreme the science may be simply a systematic codification or taxonomy of the observed information, as in classical botany. At the other extreme, such as physics, the theory is mathematical, with deductive inference used to predict facts which can then be verified by observation. As these examples illustrate, the observe-and-model cycle proceeds quite differently in different sciences; inference may be inductive and reasoning may be expressed in natural language argumentation.

If one grants the dependence of a science on natural phenomena, then mathematics would not be classified as a science, nor would studies related to computing. Although in the past, this kind of reasoning has led to describing mathematics as an "art," mathematics is also commonly called the "queen of the sciences". Based on human-produced systems, it is concerned with the development of deductive structures, many of which are applicable to the theoretical aspects of other sciences. Given the importance of explanatory theories, the identification of mathematics with the various sciences is extremely close. Therefore, it is not surprising that, in the case of mathematics, the distinction between art and science becomes blurred. In common usage, mathematics is often enigmatically included under the heading of "mathematical sciences," a phrase which, it would seem, should mean those sciences that are based on natural phenomena, whose principal explanatory theories are formulated as mathematical systems.

Computer science shares some of the attributes of mathematics. Whereas the objects of study are man-made, and while it is widely applicable in formulating theories and models in the other sciences, for the most part, computer science uses mathematical models to formulate the underlying theories. When labels are acquired through common usage, paradoxical situations are readily produced. It would appear, then, that "computer science" is a mathematical science, and moreover, that there are some sciences which use computer science heavily and which might thus be called "computer science sciences." It would be less confusing, as well as a better display of hierarchy, to say that informatics is a mathematical science and that there are certain "*informatical* sciences." But even with this formulation, we would not avoid the issue that computer science, like mathematics, is not based on natural phenomena and, hence, does not qualify, by the common definition, as a science. Before discussing this issue, though, it is useful to consider the distinctive aspects of engineering.

Engineering covers a wide spectrum of activity, ranging from highly science-oriented, creative designs to the short-term solution of problems based on a detailed knowledge of the properties and applicability of current technology. In its approach, it is often characterized as being both pragmatic and empirical, with implementation, rather than understanding, as the goal. *Applied science* is often regarded as a synonym for "engineering," which suggests that the application of existing sciences to the solution of technological problems is

the underlying rationale. Since there are usually many different ways to implement a system, one of the dominant engineering activities is the comparison and selection of alternatives. This results in a study of costs (or efficiency), which may vary in sophistication from an arithmetic calculation to a complex mathematical formulation, to find a best, or optimal, solution. It is interesting that much of computer science is concerned with the cost or efficiency of programs and algorithms, an efficiency that is usually measured in terms of the computer time or storage required. In computer science and computer engineering, this preoccupation with efficiency, in a hierarchy of algorithms ranging from procedures for arithmetic to very large systems such as air traffic control programs, makes it difficult to distinguish between the science and the engineering aspects of computer science. Perhaps the difference between the goals, understanding and implementation, is the best distinction.

Sometimes a division between the science and engineering aspects of computing is proposed on the basis of physical realization. That is, "hardware" is the engineering domain. But in an era when computer systems are microprogrammed (that is, the elementary "machine operations" are carried out by programs of more elementary steps), this, too, is an elusive criterion. In such a milieu, building a computing system involves, in large measure, writing programs, not producing new physical entities. Beyond that, people working at all levels of developing computer systems strongly believe it is not desirable to have a boundary between the machine level and higher, more macroscopic, algorithms. In short, there is currently an operational difference between computer science and computer engineering, which corresponds roughly to how close interests are to the levels of physical implementation of algorithms, or the machine level. Because of the emphasis on the cost of algorithms at all levels, however, the point of division between the two is not clear. One difficulty in making the distinction more precise is due to the absence of natural phenomena, phenomena whose exploration is the *raison d'etre* for other sciences.

Some Definitions

Many people have attempted to produce a succinct, universally acceptable definition of computer science. Since computer engineering, no matter how it is distinguished from computer science, rests ultimately on the same definition, it has not generated independent candidates for definition. The suggestion that seems to have the greatest acceptance today is one that conforms to the definition mentioned above, in that it specifies the underlying phenomena that are the objects of observation and which are explained by formulating theories. Specifically, the proposed observational base is simply the phenomena related to computers.[9] This fits the pattern of astronomy (as the study of stars) and biology (as the study of living systems); it resolves the art-versus-science enigma mentioned earlier. It is still true, however, that the phenomena specified are the result of man-made machines, not natural systems. While there are other instances where this is true (for example, library science), it does raise the question of why one should observe and model a system that has been created by human effort. To anyone who has worked with modern computers, the answer is obvious: although computers, in principle, are completely deterministic, in practice, the enormous complexity that can result from the hierarchical development of computer programs precludes a deterministic approach to many systems. The complexity of the interaction of the many executing programs often leads to situations in which the only feasible approach to comprehension is observation and analysis independent of the system of programs that were created originally. Due to the man-made nature of the phenomena under study, however, there is a *caveat.* In the effort to adopt the methods of natural science, the obvious, simple knowledge of a system must not be ignored. As an overstated example, someone who determined experimentally that computer instructions take integral multiples of a time quantum would be subject to criticism of his colleagues, since detailed knowledge of the design of the processor under study would produce this relationship directly, without experimentation.

There is another pitfall which arises from the synthetic nature of the phenomena under study. In most sciences the very existence of the domain of study is sufficient reason for observation and explanation by developing models and theories. In computer science, however, in order for this motivation to exist, there must be some rationale for generating the complex system. It makes little sense to generate a highly complex system (but one without function) and then study it in detail. That would be like producing the axioms and rules of inference for a mathematical system and then constructing and observing the many examples that can be derived from the postulated system. If the given premises are arbitrary, the system is interesting only if an analytic reduction can be obtained. Alternatively, the premises may have been prompted by some external problem, in which case there is a reason for studying the system that has been synthesized. There is little point in studying, for its own sake, complexity that does not yield to analytic organization. On the other hand, such study can be justified if there is external motivation such as the existence of the complex system in nature or the fact that the complexity arises from useful computer applications. In sciences with a given natural corpus—in contradistinction to a generated corpus—the question of whether phenomena are worth studying does not arise.

The preceding remarks have to do with another succinct characterization of computer science, one that is expressed as a comparison with mathematics. "Mathematics deals with *reducible* complexity, while computer science is concerned with *irreducible* complexity." This comparison appeals to those who have worked with large computer programs in which the strong interaction of the various parts defies attempts to separate the parts for independent analysis. Yet, on reflection, this paradoxical definition cannot be sustained. Without some kind of reduction or partitioning of complex problems, increased understanding is not possible. To be sure, the techniques of reduction in computer science are different and often are not susceptible to the analytic simplification that arises from properly posed mathematical systems. In spite of this shortcoming, however, the characterization emphasizes two salient aspects of computer science; the given computer phenomena often originate in large systems that verge on the incomprehensible, due to the interaction of the numerous parts of the systems; and the techniques for identifying, quantifying, and managing complexity are central to the various branches of computer science. Incidentally, such often-used anthropomorphic references to computers as "the computer thinks" and "the computer sees" originated partly in the perceived complexity of computer systems. Just as "thinking and feeling" are used to describe unknown human processes, the complex but deterministic results of computer programs are similarly described as a convenient, if somewhat misleading, figure of speech.

The relationship of computer science to mathematics has been argued in considerably more detail.[8] As observed earlier, the two disciplines are similar in that they deal with synthesized systems, but they differ markedly in emphasis. Mathematics deals with theorems, infinite processes, and static relationships, while computer science emphasizes algorithms, finitary constructions, and dynamic relationships. If accepted, the frequently quoted mathematical aphorism, "the system is finite, therefore trivial," dismisses much of computer science. It is true that the properties of a finite system can, in principle, be determined by enumerating the possibilities and exhaustively considering them. The usual infeasibility of such an approach, however, leads to many important analytic techniques and algorithms. These distinctions might lead one to the conclusion that the study of algorithms is the unifying thread of computer science. However, even though all the levels of the hierarchy which comprise computer systems can be interpreted as algorithms, the *study of algorithms* and the *phenomena related to computers* are not coextensive, since there are important organizational, policy, and nondeterministic aspects of computing that do not fit the algorithmic mold.

It is tempting to try to combine the various aspects of these definitions into a single statement—at the risk of producing the proverbial committee-produced, all-purpose result. Such a definition would be something like "computer science is the study of the design, analysis, and execution of algorithms, in order to better understand and extend the applicability of computer systems". Frankly, there is little hope that people in computer science and engineering will be able to agree on such a simple definition, any more than such agreement is possible in the other sciences. In any event, it does raise further questions about what exactly is meant by these general phrases. Indeed, the COSERS report is intended to be just such an elaboration. It is an operational definition, in that it describes the research done in recent years in the various computer science and computer engineering areas indicated by the present chapters. Some limited projections are also made.

This discussion came about because of the need to somehow describe the domain of the research and to argue that the collection has a coherence or overlying rationale. Before leaving this subject, it might be worthwhile to include one more approach—to list a set of general questions which computer science and engineering research is intended to answer, in the hope that the relationships between chapters will become clearer. The all-inclusive question is, What can be automated?[2] Somewhat more specific sample questions are the following:

1. Are there useful, radically different computer organizations yet to be developed?

2. What aspects of programming languages make them well suited for the clear representation of algorithms in specific subject areas?

3. What principles govern the efficient organization of large amounts of data?

4. To what extent can computers exhibit intelligent behavior?

5. What are the fundamental limits on the complexity of computational processes?

6. What are the limits on the simulation of physical systems?

7. How can processes be distributed in a communicating network to take advantage of microtechnology?

8. To what extent can numerical computation be replaced by the computer manipulation of symbolic expressions?

9. Is it possible to replace a major part of system observation and testing with automated proofs of the correctness of the constituent algorithms?

As it turns out, each of these questions fits naturally in one or more of the categories of the COSERS report. They are merely samples of deep general questions, and not the principal question in an area of research. It is clear that each of these questions is a special case of the general question, What can be automated?, and that the answers will involve algorithms and their computer implementation.

A Brief History

There are now several histories of digital computing.[6, 10] Although much of their detail is not relevant to the research emphasis of this report, it is worthwhile to consider the recent evolution in order to understand, at a macroscopic level, the patterns of motivation and response that have characterized the field. This provides a basis for predicting future developments.

The beginning of modern, digital, electronic computing is generally marked by the construction of the ENIAC in 1946 at the University of Pennsylvania, although there were some earlier design attempts.[7] While ENIAC was built to meet the requirements of computing ballistic trajectory tables and was controlled in its computational steps by a panel of wired instructions, the generality of the machine and its extension to a stored program were envisaged almost from the beginning of its development. In the late 1940s and early 50s, construction of several binary, fixed-word-size, stored-program machines was undertaken. The general design, sometimes called the "von Neumann machine," was the progenitor of the wide variety of current machines. The vacuum tube technology of the wartime years was adequate for the development of computers to this level of generality. Very quickly, as machines became operational at universities and in government laboratories, the demand for greater size and reliability spurred an intensive effort to improve the technological base, which, in turn, led to increased applicability, thus providing an incentive for more device research and technological development. Interestingly, in 1978 this cause and effect seems to be interchanged. The appearance of large-scale integrated (LSI) digital electronics such as microprocessors and random access memory chips has prompted research efforts in effective, general and specialized ways of exploiting the resultant "point" concentrations of computer power. The role of technology as the driving force commenced with the introduction of discrete transistor circuitry and ferrite core storage. It now appears that technological improvements are continuing at an increasing rate, with the development of programs and operating systems unable to keep pace. To close the gap will require substantial research.

It is popular to divide the 30-year history of digital computers into eras, or "generations," which correspond roughly to the dominant technology in use at the time. The first generation was that of vacuum tube technology. It lasted from the commercial versions of the von Neumann machines to about 1958. The discrete-transistor versions constituted the second generation and continued to about 1965. The third generation corresponds to medium-scale integration, in which a relatively small logical function is contained on an integrated circuit chip. Contemporary large computers are still in this category, although fast machines, using LSI "bit-sliced" microprocessors as the basic building blocks, are beginning to appear. As with all genealogies, branches appear and a simple generation specification is no longer descriptive. In computing, the important branches are minicomputers and microcomputers, which have a significantly higher degree of circuit integration than do the large "mainframe" machines.

Within these major divisions of time are other technological, organizational, and economic changes, which were influenced by research and which, in turn, influenced later research. The first machines resulted from the need to solve mathematically formulated scientific problems numerically. There was rapid progress in developing numerical methods and adapting well-established finite-difference formulations for use on electronic machines. The computer-oriented research of the time was dominated by numerical analysis. Not only was it a matter of reaching the original machine design goals, but the short time between machine errors and the inefficiency of input and output made these self-contained mathematical problems a good match for the machine characteristics. The input/output bottleneck arose because the computer's central processor was directly involved in the relatively slow transfer of information to and from mechanical devices which read and wrote external media.

The processor's commitment to the sequential transfer of information to or from the main storage of the machine meant that, in general, it could not carry out arithmetic steps when information was being transferred. An early augmentation of the von Neumann design was the addition of special-purpose, subordinate processors which, under the control of the central processor, could carry out the input/output transfers from a common main storage concurrently with the computational processing. These additions, called "channels," greatly improved the computational efficiency of the machines, in addition to making them more applicable to record-keeping and data-handling applications. However, the concurrency also greatly complicated programming. To use the capability, programmers had to provide for input buffering or "reading ahead"—that is, transferring input information to the main storage before it was needed in the computation task. Similarly, buffering by "writing after" permitted the output operation to continue while the central processor began another task. The proper control of the organization and timing of these buffering steps—details introduced to increase machine efficiency—placed a substantial burden on the programmer, one that was extraneous to the logic of the problem he was trying to solve. Indeed, much of the research in computer science and computer engineering has been prompted by the need to avoid this burden. While computers have the innate ability to execute algorithms of great complexity, it is obvious that their application will be artificially limited unless their description (for computer execution) can be reduced to succinct, well-organized statements free of the details imposed by the computer implementation. Historically, the subjects of operating systems, programming languages, database systems, and programming methodology originated from the desire to automate such efficiency-motivated programming, as well as to identify and implement programming languages that permit the minimal, understandable specification of certain classes of algorithms. The development of list processing techniques and recursive program formulation did much to relieve the burden of storage management and to permit the succinct statement of programs.

During the transistor generation, advancing technology made possible a reduction in the cost of high-speed (ferrite core) main storage. This development permitted larger storage capacities, making it economically feasible to simultaneously store several independent programs at various stages of processing. It was then possible to automate the buffering simply by suspending program execution when an input/output step was invoked and beginning another

stored program. This "buffering by program" (or "multiprogramming") was more readily automated than the highly problem-dependent buffering of data, although, as systems developed, the latter problem also responded to automated treatment.

The line of development discussed above has produced machines with many highly interconnected parts, which have access to common storage. The high cost of these systems, which is largely a result of the interconnection complexity, mandates well-loaded, efficient, usage. At this point in the development of computers, such "mainframe" systems are not amenable to the utilization of large-scale integrated circuits, although medium-scale integration is readily employed. Partly for this reason, minicomputers, the technological offspring mentioned above, have flourished. This type of machine, with its simpler processor and channel structure, can effectively employ LSI and benefit from the resulting cost reduction to the point where high efficiency of use is no longer the dominating criterion of system design. Microcomputers, in which the processor is on a single circuit chip, benefit further from the cost reduction that results from advanced technology; but they are simpler and slower than the minicomputers. It should be emphasized, however, that chip costs are such that it is not difficult to justify either low use or special-purpose use. The economics of the LSI components argue that they be used in a concerted way, to give large concurrent computing power but without the cost-inducing design of a closely connected switching network. This motivation, in addition to the need to connect geographically dispersed computer systems, has increased interest in computer networks.

With this "broad-brush," technology-oriented history, the current state of the art of computing is reached. Computer science and engineering research, which have contributed heavily to this history, have been motivated by the new capabilities produced along the way. Clearly, hardware systems are the basis of this development. Work in the area has been driven by the astonishing rate of device invention; but it has also resulted from the perceived need to simplify, to encapsulate detailed functions in hardware algorithms (or microprograms). And, as mentioned before, it is apparent that the recent advances in computer hardware have not yet been fully assimilated in programs and operating systems.

Applications

Over time, the programs run on computers have increased markedly in size and sophistication. But it is difficult to produce a historical sequence, since the development is a many-branched one in which each new operational program of a particular type depends on earlier experience, new insights, and new system capabilities. Many programs develop quite independently of the technology. That is, the work is limited by a knowledge of the problem area and the ability to produce an operational program that solves the information-processing problem in an acceptable length of time. Such problems are usually referred to as "applications," because their solution depends on a particular subject area, as opposed to being concerned with computers *per se*. It is a difficult distinction to make. If every application of computers is considered to be in the realm of computer science and engineering, these fields will be nearly all-inclusive. Whether the applications should be so included depends primarily on the specificity of the underlying problem. It is more reasonable to include the development of algorithms for the solution of a class of problems, as opposed to a specific problem, under the heading of "computer science". The usual property of such generality is that the dominant component of the researcher's expertise has to do with computers and algorithms. Thus the development of a program to reduce x-ray diffraction data for crystallographic analysis would depend heavily on a knowledge of crystallography and a lesser knowledge of computing. On the other hand, a program for automating the collection and reduction of data from various experimental environments would, more than anything else, require expertise in computing. A program for symbolically manipulating algebraic expressions would have many uses and would depend on sophisticated data structures and the transformation of symbol strings. Conversely, a program for generating the coefficients of a particular series depends more directly on the detailed structure of the series. It is clear that there are many shades of grey in these distinctions.

Special topics such as the generation of natural languages and the processing and manipulation of symbols are in this category of widely applicable, problem-driven work which depends largely on expertise in computing. So, too, does much of the work in artificial intelligence, where research is not limited by computers as much as by the need to understand and algorithmically emulate intelligent behavior. In this sense, that part of numerical analysis in which mathematical developments are motivated by the ultimate goal of effective computer execution is an application; the resulting algorithms are applicable to various problems. The underlying problem is the discovery of algorithms which permit the solution of mathematically posed problems of many kinds, using the necessary discrete digital representation. In fact, emphasis on computers provides a demarcation in the well-established subject of numerical analysis. If the emphasis is on the existence of solutions, and the methods are primarily those of classical real and complex variable analysis, the subject is readily accommodated in mathematics. If the emphasis is on effective computer implementation, it is closer to computer science and is sometimes called "numerical methods". This label is not completely appropriate, however, since the nature of the work is analytical.

In addition to the generally applicable, problem-driven applications so closely related to computer science and engineering, there are certain specific applications that bear the same relationship. These applications arise from large, special-purpose computer systems. Although highly specific in design, the implementation is dominated by a range of computer issues, for example,

processor organization, operating systems, digital communication, reliability, and error recovery. The implementation of air traffic control systems is in this category. Another system with the same characteristics, which is less clearly in the domain of computer science and engineering, is digital telephone switching and communication. The reason the latter, unlike the former, is more widely regarded as a problem in telephone engineering seems to be historical. Air traffic control has developed as a major application of interest, as a result of the development of large, reliable digital machines. Telephone switching, on the other hand, is an engineering specialty which predated computers by many decades, but the current implementation technology is computer oriented. It is not surprising that much of the research expertise sought for the design of new digital communication systems is emphasized by graduate programs in academic departments of computer science and engineering.

In summary, the applications of computers are myriad, and certainly most of them cannot be listed as products of computer science or computer engineering research other than indirectly through the existence of computers, programming languages, effective algorithms, and operating systems. In a similar way, the use of mathematics in various research areas is usually not considered to be mathematical research. However, there are some applications where the dominant expertise is in subjects related to computing or where innovative computing must be done to succeed. Obviously, research on computers and computer systems *per se* is in this category. Systems of general application, such as symbol manipulation or graphics, are in this category because the generality implies that there is no other dominant discipline. Lastly, large specialized systems, such as traffic control systems, are also in the domain of computing research because the central concern is the effective integration of a variety of programs and digital components.

Patterns of Publication

The much-used metaphor about the tree of knowledge—with its dormant, sometimes withering, branches, as well as those in full bloom—is as pertinent to computer science and engineering as it is to other subject areas. In the case of computing, however, the image that comes to mind is more one of an accelerated movie of plant growth than of incremental, seasonal changes. In this report the topics of current growth are emphasized. It is apparent that the length and amount of research activity varies widely from one area to the next. Partly as a rationale for the subdivisions chosen for this report and partly to obtain a view of the shifting emphases in research, it helps to look at editorial groupings perceived as being important over the years.

Formed in 1948, the Association for Computing Machinery (ACM) has been an influential professional organization for those interested in computing. As the name suggests, the early work of the association was dominated by an interest in machines, or "hardware". The first issue of the *Journal of the ACM* in January 1954 gives a brief account of the publishing activity in the preceding years. The American Institute of Electrical Engineers and the Institute of Radio Engineers formed a computer committee in 1949, and in 1951 these organizations created a professional group. With ACM, they sponsored Joint Computer Conferences, whose proceedings, in addition to those arising from ACM conferences, provided a means for research dissemination. The entry of these engineering societies (which later merged to become the present Institute of Electrical and Electronic Engineers) was regarded as an opportunity for the ACM to "direct its effort to other phases of computing systems such as numerical analysis, logical design, application and use, and to programming."

Although the journal did not list editorial areas until 1970, it was apparent that the ACM was organized shortly after its inception to accommodate research interests in areas not directly related to machine construction. Interestingly, the first issue of the journal contains articles on a programming system, an insurance premium-billing program, a new commercial machine, multiplexed analog-to-digital systems, and two articles on analog computing. Diversity was as apparent then as it is today.

Parenthetically, it is currently a matter of argument whether the machinery-emphasizing name is misleading for an association with such wide coverage. Such issues as the value of an established 30-year-old name and the computer-phenomenological basis of the work enter into the discussions. However, for the purpose at hand, the publication history provides a better understanding of the content implied by the various sublabels of *computer science* and *computer engineering*. As stated above, one of the main purposes of the COSERS report is to describe in a readily comprehensible way the research aspects of the subdivisions.

The *Communications of the ACM* was established in 1958 as a technical publication with a shorter article submission-to-publication time; this was intended to make it more responsive to the rapidly changing subject matter. In 1960 the publication listed the following editorial groupings: standards, algorithms, techniques, scientific applications, and business applications. By 1977 the list had grown to 10, although at one point there were 12, with subjects being dropped, added, and renamed along the way. The 10 groups in 1977 were artificial intelligence and language processing, computer systems, graphics and image-processing, management applications, management science/operations research, operating systems, programming languages, programming techniques, scientific applications, and the social impact of computing.

In the COSERS report the categories are: theory of computation, numerical computation, hardware systems, database systems, software methodology, programming languages, operating systems, special topics, artificial intelligence, and applications. With the exception of the first two categories, those in the COSERS report are included in the ACM list, although the division of subjects

varies slightly. Also, not all of the editorial groups recognized by the ACM are included in the present categorization. The exclusions are based on an estimate of the amount of research activity in the areas, as well as whether such computer-oriented publications are the primary outlet for publishing research results in the areas. Obviously, this raises questions about the missing two categories. While the *Journal of the ACM* has been one of the main publication outlets for research results in theory and numerical computation, by no means has it been the sole outlet. When editorial categories were listed in 1970, they included, along with several of the headings given above, automata and formal languages, combinatorial theory, the theory of computation, linear algebra, and the mathematics of computation. These topics are still strongly represented in 1979, but now they are grouped slightly differently under the headings: theory of computation; combinatorics and graph theory; automata, computability, and formal languages; computational structures; and numerical computation—topics which are subsumed in the COSERS list under theory of computation and numerical computation. The research methods used in these two areas are strongly mathematical; in fact, active research predates the development of electronic digital computers. It is not surprising, then, that there are more publication outlets and that the classic research procedure of literature review, theory revision, and subsequent publication is better established in these areas. In these terms,these fields of research are more mature than the more recently identified specialties.

Because of their early research history and the variety of journals for publishing the results of research, the question arises whether fields such as theory of computation and numerical computation would be better treated as branches of mathematics than as part of computer science. Counterarguments for such a classification are the computational motivation of the research and the fact that the contributors concentrate on journals which specialize in computer-related topics and have computer-oriented readers.

Although the creation and deletion of editorial groups in journals is indicative of the currently perceived research areas, as well as being a general response to the pressure to recognize new areas, the division is, by common admission, imperfect. For example, there is some opportunism in the selection of editors, along with the individual influences which are not truly representative of the interests of the research community. The existence of a particular editorial classification does not necessarily mean that the specialty has an active group of contributors and readers.

Another possible approach to determining publication patterns is to analyze the published articles and the references to these articles, to determine whether affinity (or specialty) groups do, indeed, exist. Admittedly, there is a possibility of circularity in this process, since published material has already been accepted under some heading and references to it may be influenced by that designation. Still, such analysis, which has been done on a limited basis, reinforces the observations one can make on the basis of the evolution of editorial subdivisions, or more simply, by looking at the recent history of computing. Limited studies along these lines[1] have verified that the theory of computation and numerical computation are areas which are most easily identified with "clusters" of inter-referenced studies, and that the much newer areas of database and software methodology do not yet have a well-developed pattern of research publication and reference. By these measures, the other COSERS categories are intermediate. The types of applications, discussed earlier, pose obvious problems in this sort of clustering analysis. To the extent that a given application is composed of work in a narrow area, one can expect a small inter-reference pattern. But listing such topics as formula processing, natural language generation, and graphical systems under a heading such as "special topics" is an artifice for avoiding many small subject headings. The comprehensive special-purpose system (for example, medical record-keeping or air traffic control) suffers from the fact that, with many such systems, communication among workers is often direct, not through journals that are generally available. An underlying tenet of journal publication is the existence of a reasonably large, unknown audience; otherwise, a system of distributed technical reports would better fill the need for communication. The digital-system type of application often draws on the research results in more specific subject areas. Primarily, it contributes the innovative integration of these results in a new system. Generally, the details of this important integrative work are of interest only to those who are undertaking the design of a similar system.

Trends

The word *trend* implies continuity over a period of time. In order of decreasing longevity, the COSERS subject areas are: numerical computation, theory of computation, hardware systems, programming languages, artificial intelligence, operating systems, database systems, and software methodology. The two types of applications, separated under "special topics" and "applications," are not readily ordered, since they are collections of topics. Again, the applications may concern particular problem areas where progress depends not so much on improving computer technology as on new knowledge of the problem; or, on the other hand, they may involve computer systems as a whole, where the integration of many aspects of computing is the dominant property. The diversity of applications suggests a case approach to the consideration of trends. Several examples are included in the chapter on applications.

Numerical computation. Numerical analysis is a venerable topic. It is a natural outgrowth of man's ancient interest in reckoning. But when large-scale computation became feasible, a branch of this field was introduced, which was less concerned with showing the existence of numerical approximations than with developing constructive, effective procedures for implementing them. For example, the precomputer topic, "calculus of finite differences," emphasized the formulation of difference operators in compact functional form. For the

most part, the computer interpretation of functions whose components are difference operators is not an effective procedure; hence, this well-developed topic is of little interest in numerical computation. The extensive work on methods for the manual computation of mathematical tables, which predates electronic machines, is more interesting. But here, too, there is no longer an emphasis on interpolation (to reduce manual computation) and sequences of operations, which produced organized and economical formats for manual recording. As a counter-trend, the analysis of the trade-off of space and time is of increasing interest and is subsumed under the general description of the complexity of algorithms.

In numerical computation an important question of possible automation concerns the extent to which physical systems can be simulated. The increasing use of simulation to obviate the creation of physical models depends, to a large extent, on numerical computation. The underlying question reduces to one of how to represent continuous physical systems, using discrete models. The discovery of effective numerical solutions to the partial differential equations describing physical phenomena is crucial in answering this question. At another level of automation, it is becoming increasingly important that algorithms be developed for selecting methods, so computer use can be extended without requiring that researchers be aware of the detailed aspects of problem solving introduced by using computers. Ideally, a dynamic analysis of an ongoing solution would select and change solution methods; indeed, such program "packages" for the solution of ordinary differential equations exist. Progress in the automatic selection of algorithms is being made in other areas. In numerical computation the development and certification of demonstrably effective computer algorithms is a natural, active research area. Along with other areas of computer science, numerical computation is concerned more and more with the complications introduced by concurrence. A lengthy, nearly independent subcomputation may be preferable to a shorter, more interdependent, equivalent step, because the first approach permits more concurrence. Research in formulating general algorithms for concurrent, or parallel, execution is increasing.

The branch of classic numerical analysis concerned with the development and analysis of algorithms for implementation on digital computers continues to be a vital part of computer science. The subject does not dominate computer science as it did in the early period. This is not due to any diminution of importance, but, rather, from the rapid expansion of nonnumerical processing. The consumers of (and often the contributors to) this work are primarily concerned with applications.

Theory of computation. The theory of computation also predates the electronic age. If the theory is taken in the sense of explanatory models—as opposed to computing devices such as those of Leibniz, Pascal, and Babbage—it originated with Turing's work in the 1930s. His seminal contribution, which was motivated by questions in logic, was to produce a definition of *computable* that agreed with the generally accepted intuitive meaning of an unambiguous, deterministic, terminating sequence of steps. Turing's definition was based on a simple, hypothetical machine—now called the Turing machine—which could read characters on a movable, extendable tape and, on the basis of what was read, select different sequences of the elementary reading, writing, and moving operations. Any algorithm that could be executed by such a machine was considered "computable" and, conversely, all others were "noncomputable". This definition was strengthened when several other independently produced definitions were shown to be equivalent. An algorithm which simulated the elementary machine itself, in much the same way that a general-purpose computer decodes instructions, is an important member of the computable class.

Although Turing's dichotomy is important, the problems of practical interest were in the computable class, and a refined categorization was needed, which would quantify "how-computable". The Turing machine and its variants were used in this quantification. Use was made of such measures as the number of elementary instruction steps and the amount of tape used as a function of given input. Alternatively, the machine could be limited to, say, two passes over the tape, thereby specifying a subclass of the computable problems. The most primitive restriction of this type is the finite automaton. In essence, it is the "control" part, the Turing machine without tape storage. The automaton has a finite number of positions, or states, which are selected from a specific history of symbol reception. These states constitute the machine's storage, or memory.

While researchers remain interested in the categorization of algorithms by the specification of such hypothetical machines, interest is growing in measures more directly comparable to real machine performance. More specifically, using independent problem size variables, such as matrix dimension or character string length, expressions for quantities such as the number of multiplications or comparisons or, alternately, the amount of time or storage required, are obtained. Such results are readily related to actual programs and also produce a more useful division of the class of computable algorithms. A well-known example of the utility of this kind of analysis is the number of comparisons needed to put n unordered items in ascending (or descending) order. The straightforward procedure of comparing the first with the $n-1$ others, and interchanging whenever an item less than the current first element is found, leads to $n(n-1)/2$ comparisons. A more sophisticated procedure, involving the repeated splitting of one unordered list into two unordered lists, in which all the elements of one list are smaller than all the elements of the other, produces a comparison count on the order of $n \log n$. For the usual values of n, the latter is clearly preferable. Measures of complexity often appear as polynomials in the size variable, n. For example, the standard method of multiplying two n by n matrices requires n^3 multiplications. This functional form introduces another interesting classification of complexity: the class of algorithms whose complexity measure can be expressed by a polynomial. The complexity of algorithms that demand more time or space is characterized by an exponential

expression in which the size variable appears as an exponent. Currently, an important research question is whether there is a complexity category whose characterization lies between these two. This developing taxonomy is important, since it distinguishes the "feasible" within the "computable." In the past there were many instances of naive attempts to solve combinatorially "explosive" problems because the complexity analysis was not known or done.

Formalisms for systematically generating sentences in a language have obvious relevance for computer science. In particular, if the generation rules are well defined, sentence recognition and parsing procedures can be produced, which can be used in translating high-level programming statements into equivalent machine-language sequences. Early work in this area emphasized the structural, or syntactic, aspects of the process. Sentence-generating rules, or "grammars," were classified according to whether context dependency existed in sentence generation, that is, whether phrases could be inserted in sentences being generated only in specific surrounding contexts. Of greatest practical interest was the context-free category. In addition, the ability to generate the same sentence with more than one sequence of generation steps leads to an undesirable ambiguity in recognizing such sentences. Thus the presence and detection of ambiguity are also matters for grammar distinction. Other language categories were created by limiting the grammars to categories where the generated sentences could be analyzed by specific algorithms. Specification of these categories has permitted the association of complexity measures with these parsing procedures, which break down sentences, or statements, into elementary parts which can then be related to machine instructions.

Recently, the emphasis in research has shifted to semantic questions which complement the now well-understood processes of generating and parsing on the basis of syntax. If the automatic verification of program correctness is to reach a useful level, the *meaning* of algorithm-describing statements must also be precisely formulated. This research goal is shared by several specialties within computer science. There is a growing feeling that there is a complexity threshold that cannot be surmounted unless programs can be verified or proved correct automatically. It is a highly introspective version of the question, "What can be automated?" The underlying problem is roughly analogous to the reliability threshold of early vacuum tube computers. Given the relatively short time between failures of the vacuum tubes, machines whose tube count exceeded a critical number were rarely operational. Similarly, the probability of logical errors in programmed system components limits the number of components that can be expected to work together.

Hardware systems. The trends in hardware development were previewed in the description of technological generations. With the usual computer pace, three major technological generations have been packed into one human generation. In this period the research areas of artificial intelligence, programming languages, operating systems, database systems, and software methodology have been created and profoundly changed. The progression from vacuum tubes and small memories to discrete-transistor and large memories, and then to integrated circuits and very large main and auxiliary memories, has dramatically changed the capabilities and economics of computing machines. Operation and access times have gone from tens of microseconds (10^{-6} second) to tens of nanoseconds (10^{-9} second), while main memories have increased from 16 thousand characters (or bytes) to 16 million bytes. Random access auxiliary storage, such as disc storage, has developed from nonexistence to 64 gigabyte (10^9 byte) or larger. If the price of these central machines is taken as being nearly constant over time, the cost of a unit of storage has dropped by three orders of magnitude, not counting the increased utility of accessing the main storage unit a thousand times more often. Beyond these major trends in computing, large-scale integration has been extended to microprocessors and memory, to produce second-generation capabilities in computers priced in the range of several thousand dollars. Apparently, "bit slice" organization will soon push these integrated-circuit machines into the speed range of tens of nanoseconds.

The reincarnation of second-generation-size machines in integrated-circuit form might be viewed as an opportunity to reapply the system organizations developed during that generation. But the large reduction in cost has dramatically changed the ground rules. Because their high cost mandated high use and centralized systems, the earlier emphasis was on creating general-purpose systems. Now special-purpose, low-use systems are justifiable and in many instances, preferable. In the meantime, the prospect of many special-purpose systems accentuates the need for the systematic programming of new systems. The low cost of such specialization also makes it possible to add redundant circuitry to increase reliability and special processing units to facilitate maintenance.

Artificial intelligence. Ever since the early days of computing, the speed and logical processing capability of electronic computers have inspired attempts to develop programs that exhibit intelligent behavior. Though "intelligent behavior" is difficult to define, and is currently understood differently by different people, there has been some convergence of view within the A.I. community as the technical requirements for the computer solution of certain classes of problems becomes better understood. To be sure, the human solution of a complex equation might be classified as intelligent behavior, while the corresponding action by a machine might not be so classified, even though both machine and man had been programmed for (learn) the process. One possible requirement is that there be something unstructured, something nondeterministic, for the solution process to qualify as intelligent. Another is that it depends on the knowledge that must be used in obtaining the solution, or on the methods used. As is often true of new fields of research, the initial efforts were very ambitious, serving primarily to reveal the myriad difficulties and highlight the subproblems that became the primary objects of study. For example, the machine translation of natural languages showed that the complexity of

the structure of languages was far more than could be accommodated by the largely syntactic and statistical procedures known at the time. Experience, however, has fostered fruitful research in developing systems which make use of a great deal more knowledge about grammar and about the subject matter of the messages to achieve much more promising results in the analysis of text and speech inputs. Various kinds of games have provided a good testing ground for developing some techniques that have proven useful for more general "intelligent" systems. The same is true of attempts to prove general theorems by computer. The mixed success achieved in the latter case has again served to highlight the lesson learned in the language analysis work — that high levels of performance require that large amounts of specific knowledge be organized and brought to bear on the problem.

As an area of research, artificial intelligence has been criticized (largely by those within the field) for these attempts to solve very general problems. The early experience with such general problems has currently led to additional research in the natural direction of smaller problem domains and the more specific use of knowledge of these domains in the solution. The mounting success of this approach — in such areas as pattern recognition, image processing, medical diagnosis, and speech synthesis and recognition — has often appeared to remove the subjects from the realm of artificial intelligence. Such "localized" research areas tend to become indistinguishable from the application areas, which are called "special topics" in the present report. Yet a number of common threads run through these applications. For example, one basic recurring problem is that of searching a large solution space and codifying the knowledge that is relevant to the solution in such a way that the use of "local" knowledge reduces the amount of searching. Even with reduction, such search can be wide ranging with unanticipated storage demands. Other such basic problems include discovering the knowledge which must be brought to bear in solving these application problems, finding computationally efficient ways of representing this knowledge symbolically and designing control strategies for accessing the knowledge when it is appropriate. Various sophisticated computational techniques developed by A.I. researchers (the earliest of which were the list processing techniques now so widely used) have greatly facilitated the programming of such problems.

Another important aspect is the use of heuristic rules of the kind humans use to solve problems. Although, in general, such rules cannot be proved effective, they often lead to solutions. Some computer scientists argue that *heuristic programming* better describes the field now called "artificial intelligence," in addition to avoiding the emotional connotations of *intelligence*. Though various alternative names have been proposed, many researchers in the field feel that the current name is appropriate, since it emphasizes its close relation to the study of human processing techniques. The field has spawned several related areas of research and is the basis for some manufacturing applications which are nearing production. Among the latter are computer controlled "visual" searching for flaws in circuit boards and some computer-controlled orientation and assembly steps. The discussion in this section also illustrates a number of other important areas of achievement, including computer-aided instruction and medical diagnosis, and applications to areas of science and mathematics. Its impact on psychology has also been far-reaching. In addition, though perhaps more significant as a matter of general interest than as an important scientific achievement, chess playing programs have reached the expert level, which is the rating just below "master."[5] A program developed at Northwestern University won the Minnesota Open in 1977, although it seems that the most recent successes are due mainly to faster computers.

Programming languages. The continuing research and development in programming languages is a good example of the process of enlarging the number of computer activities that can be automated. Before the appearance in the mid-1950s of algebraically oriented languages, the programming of computers seemed to be a "black art" practiced by a handful of people who had mastered the octal or hexadecimal representations of machine instructions and data. High-level languages suppressed this detail, greatly increased the number of programmers, and increased the complexity level of programs that could be implemented successfully. As always, the complexity barrier was not eliminated, just moved. The work continues in such directions as machine code optimization, programs to generate language translators or compilers, techniques for language extension, high-level languages, very succinct languages, language constraints to enhance program verification and comprehension, and the incorporation of more complex data structures. There are two obvious facets of these efforts, which characterize computer science and engineering in general: the dependence on other areas of research activity and the continuing presence of computational efficiency as the dominant requirement. Regarding the latter, the feeling has reemerged, with each major technological improvement, that increased machine performance makes acceptable the substantial inefficiency that might result from a very high level of programming. However, the increasing requirements of application programs at such points of technological improvement have brought pressure for the increased productivity to be made available to application programs.

It is extremely difficult to determine the proper balance between the suppression of computer-oriented detail in the specification of algorithms and the improved computer efficiency that can be achieved with detailed specification. Analogous to the construction of physical systems, it is necessary to accept some inefficiency in the interest of achieving manageable organization. In constructing multichassis electronic systems, the "efficiency" of point-to-point wiring is sacrificed for the superior organization of connecting blocks and cables. But the trade-offs in computing are often far from obvious. In principle, an extensible language may permit it to be hand-tailored for a particular problem domain, but the complexity of the language for extension can eliminate the advantage gained. There is evidence that, given language specifications, the use of a program that produces translators is a better choice. It is less complex and more efficient to develop a new specialized language, with its related compiler or translating program, than to extend an existing one. Greater knowledge of this process of compiler construction has encouraged the development of languages dedicated to specific application areas. The inefficiency of using an all-purpose language that includes many features not used in a particular application have also encouraged this development. Such unavoidable considerations of efficiency have resulted in greater emphasis on optimizing the code, or machine instruction sequences that are produced as output from a compiler. This, in turn, has raised questions about the semantics of programming languages and the design of instruction sets at the hardware, or machine, level. The interdependent nature of the various research areas is apparent in programming languages, as it is in many other areas..

Operating systems. In the hierarchical view of a computer system, operating systems are those programs that interface the machine with the applications programs. The main function of these systems is to dynamically allocate the shared system resources to the executing programs. As such, the research in this area is clearly concerned with the management and scheduling of memory, processors, and other devices. But the interface with adjacent levels has shifted, and continues to shift with time. Functions that were originally incorporated as part of the operating systems programs have migrated in succeeding systems to the hardware. On the other side, programmed functions extraneous to the problems being solved by applications programs have been automated and are included in the operating system. Perhaps this evolution can be better viewed as the insertion of levels in the hierarchy, since many of these automated functions are large and complex algorithms in their own right. The best-known example, and one that illustrates migrations in both directions, is the incorporation of virtual memory. The association of logical or virtual addresses and their assigned physical storage locations has moved from being an operating system function to being a hardware algorithm in a processor. By the same token, the programmer's task of "overlaying" the limited storage with different parts of his executing program has been subsumed as an operating system function.

The persistent quest for the extension of computer system applicability by simplifying high-level programming through the automation of such programming steps continues. The challenge is to devise algorithms that can handle the large variety of application programs with acceptable efficiency. General file system operations are examples of this trend. The increasingly important functions of file protection and authorization must necessarily be implemented by a central program or operating system. An often used analogy may help illustrate these goals. If a program could store and retrieve information from an address space labeled in a manner similar to that of a library decimal classification system, storage allocation problems would be eliminated. In such a logical system, information could be inserted or removed without the restriction of dealing with integer addresses. In fact, information in a computer is stored in fixed size blocks in a variety of storage devices. Input/output programs, with associated buffering, are required to access the various parts of the information. Insertions involving "pointers" to other locations or the restructuring of large storage regions are often needed. But it is conceivable, if not efficiently implementable yet, that application programs could use such an address space and have the operating system generate and execute all of the implied input/output steps.

Database systems. The automatic management of a storage hierarchy is obviously a central concern in the implementation of database systems as well. Indeed, the lines that separate this emerging specialty from the concerns of operating systems are not completely clear, but there are some distinguishing interests. The increasing size and decreasing cost of storage have changed the rationale for centralization. At one time interactive computing was done on a large central computer because the economy of scale was such that the least expensive computing was obtained in this manner and, also, the cost of computing exceeded the cost of communication to the central facility. The other part of the rationale was commonality. That is, the shared programs and data could be stored and maintained at a single location. Now, interactive computing can be done locally on an efficient basis and programs and data that are not changing rapidly can be stored locally since the cost of secondary storage has markedly decreased. However, the many gigabyte (a billion bytes or characters) storage that can now be attached to relatively large machines means that the large, rapidly changing databases that are the basis for corporate and institutional operation can be implemented. Thus the rationale has shifted. Interaction with a common system is important if access to a large, usually changeable database is necessary. This possibility of a central information store has motivated the increasing research in the problems of automating the updating, retrieval and security aspects of such information repositories.

Modern operating systems must have a file system for the use of both system and user programs. For the most part such files are uninterpreted by the system. That is, the files are simply a string of characters or bytes. To be sure, certain system programs may interpret the files as programs that can be executed, or lines of characters that can be changed or edited, but the more detailed interpretation as, say, personnel files, sales records, or student files is done by another level of programs. It is this very complex set of functions that is the realm of database systems.

The appropriate storage organization for a database depends upon its pattern of use. If an entire set of relatively unchanging records are always recalled in the same order, a strictly sequential organization is the best. If certain known subsets are occasionally sought, these records can be linked together in the sequential file by internal pointers. If even more flexible retrieval is likely, directories giving the location of records with specified attributes can become

part of the database structure. The database structures are many, and modern, efficient systems must be able to use different organizations while keeping the details of the various storage structures from complicating the task of the requestor of information. One of the current challenges is to devise some primitive database operations which can provide an interface between a variety of requestor languages and the many database storage formats. Such standardized "transactions" are needed to extend operating systems to accommodate the many organizational variants of databases. In terms of flexibility, the most general approaches are subsumed under the title of *relational* database systems. With this scheme any consistent, unanticipated request for a subset of the stored data can be satisfied. The necessary "relations" between different records are determined by matching values of attributes. For example, information in two different records that were identified with the same social security number could be related. If this could be done on an efficient basis for all kinds of requests, it would obviate the need for a multiplicity of approaches. Further research on the various component algorithms is needed.

Just as modern operating systems have a standardized user interface, efforts are underway to design a standarized database interface. The definition of elementary transactions is a step in that direction. As mentioned, this problem would be simplified if relational database schemes could be optimized to the point where they would have an acceptable response for all kinds, even often repeated, inquiries. However, given the variety of database usage patterns, it may be necessary to design self-organizing systems. That is, the database storage structure is dynamically modified to efficiently accommodate frequently occurring requests but it is still able to handle unanticipated requests. The necessary algorithms for storage restructuring and management are complex and under study from several different approaches.

Software Methodology. A topic that arises in all of the programming oriented specialties is the most recent candidate for identification as a subject area. Programming or software methodology, also called software engineering, is concerned with the generation of programming conventions, and constraints, which will facilitate the specification, programming, testing, verification, portability, and comprehension of large systems of programs. These goals arise from the pressing need to manage the complexity inherent in large programs and to minimize the reprogramming that is endemic in an era of rapidly changing computer technology. The techniques of interest range from the constraint of programs to readily understandable or verifiable forms, the automatic generation of test data, and the automatic imposition of program interface data. "Top-down" programming, where the macroscopic algorithm is progressively refined into sub-algorithms is an example of the first of these. The imposition of a subprocess structured program design also enhances the comprehensibility. The use of computers to record programming decisions in files and then concatenate these files to the design of other parts of the planned system avoids interface inconsistency or omissions. Beyond that, the managerial aspects of large program development are generally enhanced by such a computer-assisted procedure. It becomes possible to oversee a development in much the spirit of conventional machine production—an assembly is composed of sub-assemblies which are composed of sub-assemblies, etc. Additional declarations at the time of programming can, in some cases, be automatically utilized to generate trial data values for testing which are more comprehensive than those usually produced by human means.

Automatic program verification is an important but elusive goal. In essence, a program is accompanied by an independent description, or set of assertions, at various entries in the program. It is possible, in some cases where the programming constructions are suitably restricted, to prove that the intermediate and final output assertions logically follow from preceding assertions and the intervening program statements. The difficulty of extending such approaches to programs of significant size and complexity is as great as it is desirable. As with theorem proving in general, it seems that interactive schemes which involve human surveillance of the successive assertion proofs may well be the most effective approach.

Software engineering is a specialty in its infancy, and it pervades many of the research goals of computer science and computer engineering. It is clearly motivated by the need to increase the level of manageable complexity.

Interrelations. It is evident from even a casual look at the chapters of this report that the research practitioners in the various subject areas are concerned with many of the same goals but from a point of view that is suggested by the area name. The basic question of "What can be automated?" can be paraphrased as "What are the limits, imposed by complexity, on the application of computers?." Recurring themes are: the efficiency of computation; the succinct, comprehensible statement of algorithms free of detail extraneous to the problem; the automatic choice of algorithms for applications and computer facility management; exploitation of concurrent computation; and the verification and testing of programs. These concerns can be viewed from the perspective of major applications as well. Several illustrations appear in the chapter on applications.

The Hierarchical World

Computer systems are almost always hierarchically structured. Many of the research specialties correspond to a level in this structure—with sizable overlaps of neighboring levels. Thus hardware interfaces with operating systems which, in turn, interact strongly with applications, as well as supporting the functions of database systems and programming languages. The latter two are used directly by all types of application programs. Computer science theory attempts to construct and classify algorithms that are relevant at all levels, while

the concern of software methodology with the process of implementing programs is similarly comprehensive.

This description is reminiscent of the popular histories of science and discovery, which start from the stratification and specialization of nature and proceed through social structures to the multi-level organization of knowledge. The progression is from atoms, to molecules, to proteins, to living systems, to the diversification of species, to human communities, and ultimately to the hierarchy of knowledge and science. The remarkable accomplishments at the highest level would not have been possible without the many supporting strata below that level. If every mathematical theory had to start with, say, the first principles of counting, progress would have been glacial. The hierarchy implicit in modern computing systems may well be the most complex system yet devised by man. This synthesis, which has been a collective effort, displays many of the evolutionary features of natural systems. Programming languages fall into disuse; systems change under the pressure of technological progress and obsolescence. Here, too, the accomplishments and applications at the highest level would have been minimal, for example, if every program had to be expressed as the state of the transistors on each integrated circuit chip. Although the computing hierarchy has evolved well beyond such a primordial point, when it is compared with the historical development of knowledge it is probably no further along than, say, early Renaissance. In any event, computing and social development are now inextricably related. Man's continuing social evolution and augmentation of knowledge will be shaped largely by his ability to understand and manage the multiplying complexities of a crowded, interdependent world. Raising the level of manageable complexity by continuing to develop computer and communication systems, which can be exploited rapidly and reliably, is imperative if these problems are to be understood and solved.

Prognostication

For the most part, this report avoids the temptation to be oracular about the details of the future computer age. Indeed, the underlying premise is that the basic question of what can and what should be automated can only be answered with study and research. Nonetheless, the potential of computers is so well known that many plans are being laid for systems and applications that will change operational behavior at levels that range from national to personal. In the comments that follow, a few of the motivations for plans in defense, information retrieval, industrial production, and individual consumption are given.

In a world armed with increasingly sophisticated weapons, military communications and control systems must be developed which coordinate many computers with the necessary high speed and reliability and yet present, in a readily comprehensible form, the information necessary for human decision.

With knowledge doubling every seven years, the limitations to progress may not be knowledge itself, but timely access to what is known. As an example, health care would be improved if practitioners had better access to medical literature and recent research results. Planning is underway to establish an accessible national medical information repository using large computers.

The economic pressure for increased productivity is very difficult to satisfy. In areas such as agriculture and electronics, where the country enjoys a leading position, retention of this advantage will require even more coordination and optimization—for which computers are already heavily used. In industrial production, the need for highly skilled workers, or artisans, is increasing. The use of computers for complex but routine tasks, like wiring and system installation, can effectively amplify the skills of such people. They are able to devote more effort to the judgmental aspects of their tasks rather than the routine mental activity that accompanies many technical jobs in an advanced society.

The burgeoning recordkeeping arising from increasing interdependency imposes a similar mental burden which can be lightened by further use of computer-communication systems. In what has been called "office automation," inter-office communications are augmented to the point where information from databases is directly available, and messages are automatically routed and stored. Text for communication is stored for editing and revision, and is displayed in final printed form as needed.

On the consumer side, the era of communication from residential television is at hand. Communication is rapidly moving to digital signals with computers for switching and message control, and fiber optics effectively increase the amount of information available to consumers. The much publicized microcomputers introduce the possibility of optimizing residential systems. The most obvious application is the timed, environment sensing control of energy consuming devices such as heaters, heat pumps, and furnaces.

Obviously this brief sample of societal *desiderata* can be greatly extended. Such an extension might run the risk of overwhelming the reader with the number and complexity of the problems and inducing a longing for a return to a simpler era. Return is impossible even if it were truly desirable, but, importantly, computers and communications do offer the possibility of automatically handling, without undue human burden, the complex tasks that persistently arise. However, it is vital, in the literal sense of the word, that social and technical bounds on such automation be continually developed, refined, and explained.

REFERENCES

1. Private communication from Gerard Salton.
2. Attributed to George E. Forsythe.
3. "Electronics," *Science* **195**(4283) (March 18, 1977).
4. *Scientific American* **237**(3) (September 1977).
5. H. Berliner, "A Chronology of Computer Chess and Its Literature," *Artificial Intelligence* **10**(2), pp. 201-214 (April 1978).
6. H. H. Goldstine, *The Computer from Pascal to Von Neumann,* Princeton Univ. Press (1973).
7. J. Howlett, N. Metropolis, and G-C. Rota (editors), *A History of Computing in the Twentieth Century,* Academic Press (1979).
8. D. E. Knuth, "Computer Science and Its Relation to Mathematics," *American Mathematical Monthly* **81**(4), pp. 323-343 (April 1974).
9. A. Newell, A. Perlis, and H. Simon, "Computer Science (letter to the editor)," *Science* **157**(3795) (September 1967).
10. B. Randell, *The Origins of Digital Computers,* Springer-Verlag (1973).
11. C. P. Snow, "Government, Science, and Public Policy," *Science,* **151**, pp. 650-653 (February 11, 1966).