Computer Science: A Conceptual Framework for Curriculum Planning

Saul Amarel
Rutgers University, New Brunswick
New Jersey

Two views of computer science are considered: a global view which attempts to capture broad characteristics of the field and its relationships to other fields, and a local view which focuses on the inner structure of the field. This structure is presented in terms of the kinds of knowledge, problems, and activities that exist within the discipline, as well as the relations between them. An approach to curriculum planning in computer science is presented which is guided by the structure of the field, by the fact that change is an important feature of the situation, and by the expectation that computer science will continue to increase its working contacts with other disciplines.

Key Word and Phrases: computer science, curriculum planning, education
CR Categories: 1.0, 1.52

1. Introduction

Computer Science is a young and rapidly expanding discipline. It began to assume an independent identity in American universities in the late fifties as the digital computer (scarcely ten years old at the time) was penetrating government, science, technology, and business, at extraordinary rates; and as a body of experiences and concepts was emerging that pointed to the intellectual and social significance of a computer oriented culture.

In order to clarify the nature of computer science, it is useful to consider two views of the field: a global view which attempts to capture broad characteristics of the field and its relationships to other fields, and a local view which focuses on the inner structure of the field. The global view is concerned with the kinds of objects, phenomena, and concepts that form the domain of discourse in computer science, and also with the pattern of relationships between this domain and other domains of study. The local view is concerned with the kinds of knowledge, problems, and activities that exist within the discipline, as well as the relations between them.

A clear conception of what computer science is and how it relates to other disciplines is necessary for the development of a sound computer science program in a university. Both the global and local views of the field are needed in order to guide (follow) the course of the discipline in the present highly dynamic stage of its development. It is important for the discipline not to grow in isolation, but to actively seek and strengthen intellectual bonds and working contacts with other disciplines. It is also crucial for computer scientists and educators to keep examining the goals and values of computer science in order to maintain a rational basis for detailed academic planning. A running discussion of goals and approaches is also necessary in order to shape and consolidate the identity of computer science not only vis-a-vis "outsiders" but also within the "computer community."

Thus, the development, evaluation, and justification of academic programs in computer science can best be carried out in a framework that combines a view of the field which highlights goals and points of contact with other fields, and a view which focuses on the structure of current subject matter within the field.

2. The Global View

Computer science is concerned with *information processes*, with the procedures and information structures that enter in representations of such processes, and with their implementation in machines. It is also concerned with relationships between information processes and classes of problems that give rise to them, and with general methods for solving problems with the help of information processing machines.

The *computer* and the phenomena surrounding it are the main objects of study in computer science. Work in the field is focused on the structure and operation of computers, on the principles that underlie their design and programming, on effective methods for their use in different classes of information processing tasks, and on theoretical characterization of their properties and limitations. Also, a substantial effort is directed into explorations and experimentation with new computer systems, and with new domains of intellectual activity where computers can be applied.

The main concern in the field is with *man-made* information processes and with systems that are designed to achieve desired *goals*. An essential part of this activity is the understanding of the variety of possible elementary processes and possible schemes of combination that can be used in the design of information processes. The study of information processes that occur in nature can contribute substantially to such an understanding. To specify an information processing system that satisfies desired goals, we also need a rationale and a set of methods for choosing from the space of possibilities the one which is most appropriate for the attainment of desired goals. To physically implement a desired information processing activity, we need, in addition, convenient and flexible schemes for transforming the abstract specification of an information process into an operating machine. The stored program digital computer provides such a scheme, and this is the main reason for its centrality in the field.

There exist in nature important information processes that are of great interest to computer science: for example, perceptual processes in the nervous system or cellular processes that direct protein synthesis via interpretation of genetic information. An understanding of these processes enriches the pool of basic concepts and schemes that are available to computer science. In recent years, research in information processing in the nervous system has stimulated the study of new logical building blocks for computers (e.g. threshold logic [27]). In turn, application of computer science models to neurophysiology has resulted in theoretical insights into important neural structures (e.g. visual pathways, reticular formation [23]). Similarly, automata theory, a theoretical component of computer science, is beginning to provide promising models for the study of processes that transcribe, translate, and control genetic information in cells.

The central role of the digital computer in the field is due to its near *universality* as an information processing machine. With enough memory capacity, a digital computer provides the basis for embodying any information processing machine, provided that the task to be performed by the machine can be specified in the form of a program that can be stored in the computer memory. In addition to a memory, the other key component of a computer's structure are a processing unit wherein a set of elementary information processes are carried out, and a controller that directs the pattern of operations in the processing unit in accordance with the program which is stored in memory. The stored program digital computer enables us to conveniently build any number of information processing machines where operation and structure can be easily separated from each other. A program in the computer memory represents the operation of a specific machine, and the controller together with the processing unit provides the structural elements that carry out the desired operation.

In addition to the fundamental property of universality, which is inherent in the digital computer scheme, there is another reason—of pragmatic and methodological significance—for the centrality of digital computers in a science of information processing and problem solving. Today's computers are concrete physical devices that carry out complex information processes in reasonable times (minutes and hours rather than days and years). This way they provide a tangible focus for ideas that can be developed and tested over a scale of time which matches well the normal human attention spans. Furthermore, they stimulate the creation of new concepts, and they permit a sustained and critical examination of concepts and of models through physical experimentation. In many complex systems where the number of variables is large and their interdependence is high, computer runs provide the only possible method for gaining reliable insight into the phenomena involved. In his 1968 ACM Turing Lecture [18], R. W. Hamming speaks forcefully of the centrality of the computer in the field, and he points out that without the machine almost all of what we do in computer science would become speculation similar to that of the Middle Ages Scholastics.

There are two major components of activity in computer science: one oriented to system *synthesis*, exploration, and innovation; and the other oriented to *analysis*, search for fundamental principles, and formulation of theories. A continuous interaction between these two components is essential for a vigorous rate of progress in the field. The activities in the synthesis-oriented component have a strong pragmatic flavor, and they have been largely responsible for major advances in computers and for the great diversity of areas in which computers are being applied. The analytical work in the field promises to provide conceptual guidelines for more efficient and more powerful designs and uses of computers.

Experimental work in computer science requires extensive use of computers, and it often stimulates new developments in computer design and utilization. Typical experimental activities may involve the development and evaluation (via computer simulation) of a new memory configuration for computers, or the development of a language for convenient communication with the computer in problems of urban planning. In these activities, workers in computer science are heavy consumers of computer resources. A close coupling with a computer center is necessary for success in this type of experimental work. In turn, experimentation in computer science tends to strengthen the tools and facilities of the computer center, and to raise the general level of its capabilities.

Theoretical work in computer science relies on several branches of mathematics and logic. A typical theoretical problem may focus on the characterization of a specific class of computer procedures (for example, algorithms for syntactic analysis in a class of computer languages), the analysis of their structure, and the establishment of bounds on the storage space and time that they require for execution. Another type of problem may seek to demonstrate that the utilization of a given restrictive rule in a computer procedure for proof finding in the first order predicate calculus does not affect the completeness of the procedure. In these examples, the objects of discourse in the theory are *computer procedures and their properties*. Just as mathematics is used in physics to develop theories of certain physical processes, mathematics and logic are used in computer science to develop theories of information processes. Furthermore, mathematics and logic are likely to receive stimulation from their use in computer science, as Analysis was stimulated by Mechanics.

There is a closer bond between computer science and mathematics: a common concern with formalism, symbolic structures and their properties, and emphasis on a set of general methods and tools that can be used in a great variety of situations. These connections between computer science and mathematics, as well as connections with other closely related disciplines, are well described by A. Newell in a memorandum to the Committee on Support of Research in the Mathematical Sciences [28]:

> Computer science shares with mathematics a concern with formalism and a concern with the manipulation of symbols. It also shares with mathematics the role of handmaiden to all of science and technology. It shares with electrical engineering the concern with the design and construction of information processing systems that accomplish ends. It shares with all of engineering a concern with the process of design, considered as an intellectual endeavor. It shares with linguistics a concern with language and communication. It shares with psychology a special concern with forms of information processing that result in intelligent behavior, broadly viewed. It shares with the library sciences a concern with how to store and retrieve large amounts of information, either as documents or as facts.

The *physical structure* of a computer (the hardware) consists largely of electronic building blocks that implement a variety of switching, storage, and communication functions. In the logical design and system design of computers, the concern is with the choice of hardware building blocks and with their local and global organization in the light of given operational requirements. These design areas are important subjects of study in computer science. They also have strong points of contact with work in electronics and in solid state physics which is concerned with the physical aspects of computer hardware. Frequently, new developments in computer electronics and in manufacturing technology impose new requirements on logical design. Also, new concepts in logical design and new modes of computer utilization are likely to stimulate new developments in computer hardware. Thus computer hardware is a natural domain of cooperation between computer science and electrical engineering.

One of the important reasons for the concern of computer science with the *process of design* is due to the strong synthesis-oriented component in the discipline. Computer systems are highly complex man-made entities that are intended to operate effectively and efficiently over a wide range of problem environments. The planning and design of these systems can be guided only in part by systematic knowledge, and it involves many of the uncertainties and the methodological difficulties that people face in such areas as city planning, the design of communication or transportation networks, or the planning of educational systems. Thus advances in methodologies of design for complex systems are of great relevance to computer science.

There is a more fundamental reason for a close coupling between computer science and a science of design. It comes from the concern of computer science with the intellectual processes of problem solving and goal-oriented decision-making that are present in design processes. The concern here is not with understanding a given state of nature but with understanding a process that involves conceiving alternative states, evaluating the alternatives, and deciding on a course of action in the light of given goals. Complex symbolic representations of situations, of desired conditions, and of relevant facts, processes of search and decision-making, procedures for deductive reasoning, and methods for evaluating merits of partial solutions are all objects of study in computer science; and they are also key components of design processes.[1]

The development of computer science has been strongly stimulated by demands for the *application of computers* in a wide variety of new areas. There exists a "moving front" of computer applications that started its journey in the late forties and has traversed since then a path which goes roughly as follows: numerical calculations, processing routine business data, assembly and compilation of programs, managing large data bases, modeling and simulation of complex systems, and performance of complex diagnostic and design tasks. In parallel with this movement of the applications front, there has been an evolution of machine designs, programming schemes, computer languages, types of algorithms, and approaches to problem solving. More fundamentally, the conception of the computer has evolved from that of a rapid calculator to that of a flexible symbol manipulator capable of performing fairly demanding intellectual tasks. The moving front of computer applications is an essential domain of activity for workers in computer science. The challenges presented by new applications and the constructive attempts to meet them are key factors in the growth and maturation of the field.

There is a missionary-exploratory attitude in computer science as it interacts with other disciplines at the moving front of computer applications. Much of it is motivated by the interest in identifying significant new classes of problems and phenomena in information processing. There is also a strong interest in testing the reach and limitations of current ideas and techniques in the field.

At present, new computer applications are being explored in almost every discipline: in mathematics (investigations of finite topological spaces, coset enumerations); in logic (decision procedures, methods of deduction); in statistics (properties of stochastic processes); in physics (control of experiments, bubble chamber analysis, organizing new bodies of experimental data, simulation of plasmas); in chemistry (organic synthesis, interpretation of spectrograms, formation of structural models for large molecules); in different branches of engineering (computer-aided design, simulation); in architecture (graphic representations, design); in urban and regional planning (transportation networks, design, simulation, optimization); in the social and behavioral sciences (management of data bases, modeling and simulation); in education (computer-aided instruction, resource scheduling); in biological sciences (organizing experimental data bases, study of models, morphological analysis of cells); in medicine (managing data bases, diagnostic processes, psychiatric interactions); in law (organizing and managing statutes); in language and literature (phonology, grammatical structures, studies in style); in the arts (music composition, pattern analysis).

An important application for computers, which is of special interest to computer science, is the design of more powerful, efficient, and easy-to-use computer systems. Developments in system software and in time-sharing systems are in this area. The use of computers in the study of computers and in their improvement is a powerful means for gaining the knowledge and insights that computer science seeks, while at the same time the field is being bootstrapped.

In most of the activities at the frontier of computer applications, there is collaboration between workers in computer science and specialists in the respective discipline. Through this type of collaboration, the contact between computer science and other "computer-consuming" disciplines (essentially all disciplines that involve any intellectual activity) is likely to continue for some time.

One of the most significant effects of computer penetration in a discipline is an increase in clarity of the concepts, problems, and methods in the discipline. In some cases, major changes in point of view take place that affect, in a fundamental way, the theoretical frameworks in the discipline. Such is the case in parts of psychology, where models of cognitive processes are being developed in the image of computer processes [35].

From the experience of work with different computer applications, a body of knowledge is growing which includes properties of different classes of problems and solution methods, and characteristic patterns of information processes. Theoretical questions are beginning to emerge about methodologies of inquiry, and about *possible modes of transforming knowledge into decisions* during problem solving processes. The study of these questions is beginning to create points of contact between certain classical parts of philosophy (nature of knowledge, methodology, epistemology) and computer science [6, 30].

In bringing a computer to bear on a "real life" problem, the problem must be formulated in a form that can be accepted by the computer, and furthermore, computer methods must be available for approaching the problem. Bridging the gap between a real life problem and its representation within a system wherein the problem can be studied and solved is a complex intellectual activity which is characteristic of much work in applied mathematics. This involves the formulation and study of models, the creative choice of viewpoints

[1] A case for the importance of a science of design in a world filled with complex man-made systems and a discussion of the close kinship between such a science and computer science are convincingly presented by H. Simon [41].

for approaching a problem, and the development of broad methodologies for problem solving. In these areas, there is strong overlap between *applied mathematics* and computer science.

3. The Local View

The concept of a *procedure*—an algorithm—is of primary significance to computer science. A computer program is a procedure in a computer language that expresses a method of carrying out a desired information processing function. The language used to formulate the procedure must be "understandable" by the computer, i.e. any statement in the language must be such that it can be appropriately interpreted by the computer.

A typical information processing function is to find solutions for a given class of problems. Examples of such problems are: invert a matrix, sort given data, translate between a pair of computer languages, find the optimal location for a regional highway, decide on a medical diagnosis, establish whether a given statement is a logical consequence from a given body of premises.

Communication of a procedure to computer hardware must take place in "machine level" language, but such a language is usually too cumbersome for expressing concepts that enter in the formulation of procedures for many problems of interest. At present, there are several application areas where a procedure can be formulated in a "high level" language which is designed for convenient communication of concepts and operations that are characteristic to the area. For each high level language, there are translators into machine level language that take a procedure from a form which is convenient to man to a form which can be directly handled by the machine.

The intellectual effort that goes into the formulation of a procedure in a given language, regardless of the level and conceptual power of the language, consists of finding a combination of steps that can be expressed in the language and that collectively represent a correct method of solution for a given class of problems. Usually, one looks for a combination of steps which is optimal in some well-defined sense.

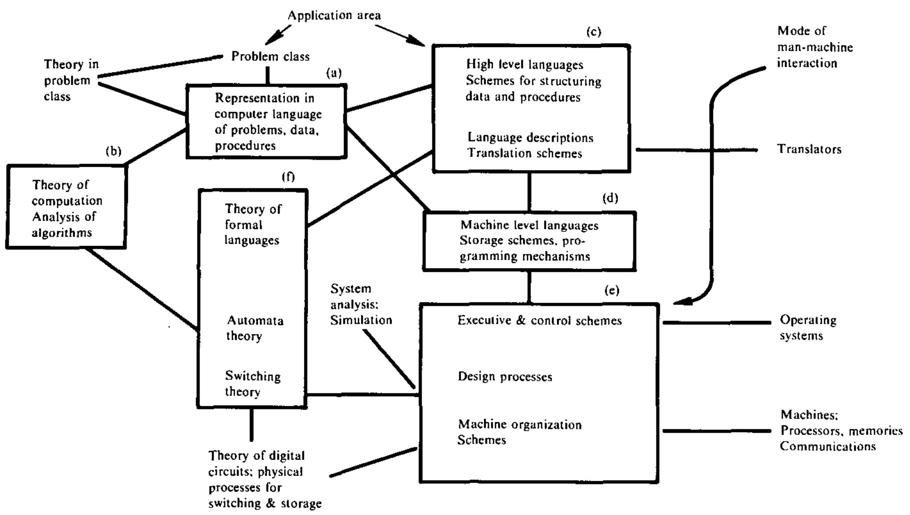
An important part of the procedure formulation effort involves analysis and validation of proposed procedures. A body of theory is slowly emerging for guiding these evaluation processes. A theoretical treatment of a procedure's validity is more likely to succeed when the procedure is formulated in a language where the essence of the method expressed by the procedure can be clearly seen; the situation becomes much more difficult when the formulation language is close to machine language. In present day practice, the evaluation of procedures relies mainly on experimental tests on a computer. Considerable effort is now being directed to experimental methods and facilities for testing, validating, and debugging procedures. An alternative approach to the formulation and implementation of procedures is also emerging which attempts to avoid the painstaking process of debugging. Instead of separating the processes of program formulation and of program debugging, guidelines for program composition are being developed such that, if properly applied, they provide a priori insurance of program correctness. Dijkstra's work on structured programming [8] is an important contribution in this area.

The conception, formulation, computer implementation, analysis and evaluation of procedures for a broad variety of problems constitute a major part of the activity in computer science. Closely associated with these activities are efforts to develop schemes, means, and tools for building and executing programs—such as languages, major principles for structuring procedures, programming mechanisms, computer organizations and design aids to facilitate these efforts. In addition, a significant amount of effort is directed to the design of advanced systems (software and hardware). All these activities have important connections with several theoretical efforts in the field—some in application areas, and others in the analysis of algorithms, in formal languages, automata theory, switching theory, and system analysis.

In Figure 1 we show diagrammatically the major areas of activity in computer science and certain relationships between them. The activities that are grouped together in boxes are assumed to be closely related. A line linking two parts of the diagram indicates close conceptual coupling between the parts. This means that the linked parts (they stand for activities or groups of activities) have closely related subject matter, and the state of knowledge of the one is highly relevant to the state of knowledge of the other.

We shall discuss below the activities that enter in the boxes of Figure 1. Each box is discussed under a label, (a) to (f), which also marks the box in Figure 1. The grouping of activities in boxes was chosen mainly for convenience of exposition. The assignment of labels, however, imposes an ordering on the boxes which reflects a specific approach to the setting of priorities in a computer science program. Our emphasis is on the relationships and mutual impact of computers and application areas. This implies building an academic program on a strong foundation of activities in problem solving, computer procedures, languages, and associated theories. In addition to these activities, work is needed in other key areas of the field, especially in computer mechanisms, in system designs, and in related theories. This work, however, should grow in an environment which is rich with significant and challenging computer applications. Clearly, this approach represents one specific point of view, and others are possible. At any rate, it is important for curriculum planners to clarify their point of view about priorities before proceding with detailed programs.

Fig. 1. Structure of major areas of activity in computer science

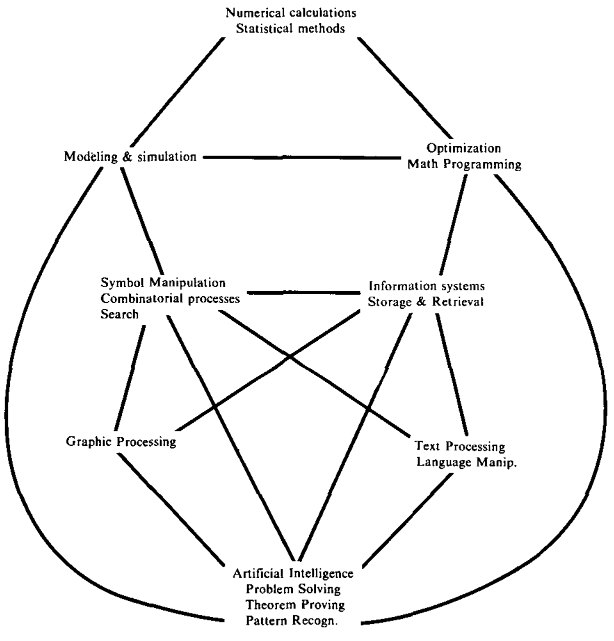

Application area
Theory in problem class
Problem class
(a)
Representation in computer language of problems, data, procedures
(b)
Theory of computation Analysis of algorithms
(f)
Theory of formal languages
Automata theory
Switching theory
Theory of digital circuits; physical processes for switching & storage
System analysis; Simulation
(c)
High level languages Schemes for structuring data and procedures
Language descriptions Translation schemes
(d)
Machine level languages Storage schemes, programming mechanisms
(e)
Executive & control schemes
Design processes
Machine organization Schemes
Mode of man-machine interaction
Translators
Operating systems
Machines; Processors, memories Communications

(a) Representations in computer language of problems, data, and procedures in an application area. The main problems in this area are to find solution methods for classes of problems in different domains of application, and to formulate them in a suitable computer language.[2] As mentioned previously, it is difficult to see any limits to the possible extent of computer applications in any area of intellectual activity. Computers will be used in all disciplines, and for a great variety of tasks. From the point of view of computer science, it is useful to classify applications not by discipline or subject matter, but by the kinds of problem solving methods, information structures, and procedures that are characteristic of the problems in the application.

In Figure 2 we show a set of eight major application areas that have achieved a reasonable degree of distinct identity from the viewpoint of computer science. In the figure, we have organized the application areas into a connected structure. The connections are intended to denote conceptual closeness between the areas in terms of solution methods, types of data and procedures, and schemes of processing. At the top of the diagram we have three areas that are principally concerned with numerical processing: Numerical Calculations and Statistical Methods [22]; Optimization and Mathematical Programming [25]; and Modeling and Simulation [43]. The remaining five areas in the bottom of Figure 2 are primarily concerned with nonnumerical processes of a variety of types: Symbol Manipulation, Combinatorial and Search Processes [40]; Information Systems, Storage and Retrieval [3, 38]; Text Processing, Language Manipulation [14]; Graphic Processing [17]; and Artificial Intelligence: Problem Solving, Theorem Proving, Pattern Recognition [9, 29, 36].

In general, every area has some points of contact with the other areas; our connections in the diagram are indicative of the major couplings only.

Fig. 2. Application areas from the viewpoint of computer s cienc


Numerical calculations Statistical methods
Modeling & simulation
Optimization Math Programming
Symbol Manipulation Combinatorial processes Search
Information systems Storage & Retrieval
Graphic Processing
Text Processing Language Manip.
Artificial Intelligence Problem Solving Theorem Proving Pattern Recogn.

[2] There is an extensive literature in this area, mostly organized by domains of application. General introductory treatments may be found in [12, 16].

Now a specific class of problems in a discipline may contain elements that are characteristic of several of our application areas. As the scope and complexity of "real life" computer tasks continue to grow, they will become less and less distinguishable in terms of content in one or more of our areas. In the seventies we can expect several major computer applications that require concepts and techniques from all the areas shown in Figure 2. For example, consider a medical application where the objective is to build and manage a data base about a class of diseases and about a group of patients, and furthermore the data base is to be used for medical education and for diagnosis. From the viewpoint of computer science, this type of application presents problems in (at least) the areas of Information Systems, Artificial Intelligence, and Statistical Methods. A complex task of design (say, in the context of regional planning) may pose important problems in all eight application areas. Yet components of the large task may be strongly identified with specific application areas, and thus the knowledge in each area may be effectively utilized in the problems of each component.

The dissection of a large task into parts, in a manner that would permit the best utilization of existing knowledge for the treatment of each part, requires deep knowledge of the concepts and techniques in each of the major application areas. This also presupposes the ability of taking the "real life" task, decomposing it in various ways, and representing it in a manner which is appropriate for computer processing. In the seventies, we will need an increasing number of people with this kind of knowledge and such skills.

Consider now a problem domain in a given application area. Suppose that there exist high level languages in this area, and also schemes for structuring data and procedures. Suppose further that there exists a body of theory which is relevant to the problem domain. Several types of procedure writing activities are possible: the formulation of some effective procedure in an appropriate computer language where none existed previously; the formulation of a procedure which is appreciably *better* than existing ones (in a sense of "better" which is intrinsic to the logic of solution in the given problem domain); and the development of a procedure that responds to a class of problems which is much *broader* than one or more classes for which specific procedures exist already. In all these cases, the essence of the effort is to find a method of solving problems in a given domain in terms of the knowledge and concepts (the theory) in the domain. Most of the contributions to method finding and procedure writing in a problem domain are likely to come from specialists in the domain (discipline) in cooperation with workers in computer science. The relative contribution of computer scientists is especially significant in the initial stages of computer penetration in a problem domain. An important type of contribution consists in the development of fairly general computer methods that can assist in the formulation of problems, in the construction of solution procedures, and in the utilization of new knowledge about the problem toward the improvement of solution procedures. Recent studies in artificial intelligence are relevant to work in this area [34, 1].

A domain to which computer scientists have been contributing most directly consists of problems in computer system design, both hardware and software. This includes problems in computer-aided design of computer hardware, problems in design of language processors (such as translators from high level language into machine level language), and approaches to the formulation of procedures for controlling the operation of a computer and for managing its resources. At present, there is vigorous activity in this domain; the level of activity is likely to rise in the next decade.

(b) Theory of computation; analysis of algorithms. There is a body of theory from mathematical logic which is concerned with computability and recursive functions [7]. The emphasis in this area has been to establish whether there exist processes that can compute certain broad classes of functions. The concepts and formalisms of this theory have contributed general insights and clarifications to problems of computation, and they have provided useful frameworks for formal approaches in computer science [32]. However, the distance between the concerns of the theory and the main problems of computer science remains great.

There are beginnings of a movement toward the study of particular classes of algorithms and their properties; for example, characteristizations of the complexity of classes of computations, and of relationships between the structure of a class of algorithms and a class of computational tasks (e.g. the work on Perceptrons by Minsky and Papert [33]). There are also beginnings of significant research on whether an algorithm ever terminates for inputs satisfying certain conditions, on the equivalence of two algorithms, on transformations of algorithms that preserve equivalence, and on the semantics of languages in which algorithms are formulated (e.g. the work of McCarthy, Floyd, and their colleagues at Stanford [31, 11]). These theoretical developments are of great importance for activities of procedure construction, debugging, and evaluation.

As computer science matures, one of its important objectives is to develop models and theories of computation that are responsive to the major design problems in the field. This requires borrowing some fundamental ideas and approaches from work in mathematical logic, but also fashioning and extending the ideas in a manner which is most appropriate to the field.

(c) High level languages for different application areas; schemes for structuring data and procedures; language descriptions; translation schemes. Given an application area, the major requirement for a high level language in which solution methods (procedures) are to be represented is the convenience and naturalness of expression for the concepts that are used by a specialist in the area in describing his methods and the knowledge which is relevant to them.

Many high level languages have been formulated to date in a variety of application areas [13, 39]. Many more are likely to emerge in the next few years. The development of a high level language usually involves contributions both from specialists in the area and from computer specialists. In past developments, the major contributions have been made by computer people. Progress of current work on convenient methods for describing high level languages may bring us to a point where the formulation of a high level language will be mostly done by specialists in the area. The development and study of schemes for language description are an active area of computer research. Innovations in this subject will contribute to the transfer of specific software design activities from the computer area to the different application areas. By facilitating this process, computer science will be making strong contributions to the effective utilization of computer power in a broad range of problems.

The formulation of a high level language is central to the entry and strong penetration of computers in an application area. Even in well-developed areas, from the viewpoint of computer usage, a high level language provides a flexible means for writing and testing complex procedures within a reasonable period of time. This is especially relevant to the development of system software within the computer industry. At present, considerable effort is being directed to the development of software implementation languages in which procedures that are major parts of system software (e.g. compilers, file processors, schedulers) can be conveniently written [44, 46].

Another kind of contribution which is also directed to strengthening means of expression for solution methods is the development of basic procedure schemes. While a high level language provides means for expressing in detail the individual steps of a method, the development of procedure schemes provides guidelines for expressing solution methods at the global level. The work of Knuth, *The Art of Computer Programming*, is a significant step toward organizing existing knowledge in this area [24].

An important scheme of procedure construction consists of a structured data base and a control procedure that processes input data in accordance with the data base. Many pieces of software have already been fashioned according to this scheme. A prominent example is the syntax-directed compiler [10] where a description of a high level language is stored in a data base, and a control procedure accepts statements in that language and translates them into machine language in accordance with the language definitions in the data base.

Procedures that are built in the form of a data base and an associated control procedure are beginning to appear in areas of computer aided design, medical diagnosis, and computer assisted instruction. In the coming years we are likely to experience an enormous growth in the number of procedures of this general type, especially as management information systems and computer utilities will begin to emerge. In the development of such procedures, a major part of the effort will be directed to the formulation of languages in which information can be described for a data base, and to the preparation of the specific content of a data base in a given area. The development of languages for data bases is another major area of cooperation with other disciplines.

It is reasonable to expect that the major method of expansion of computer usage into new application areas will be via the development of convenient frameworks and forms that specialists can use to express their knowledge, work rules, and methods, in a manner that can be "understood" by computers. High level languages and broad types of procedures that can easily accept and use new knowledge are major contributions to these problems of flexible man-machine interface.

(d) Machine level languages; storage schemes and programming mechanisms. A language at the machine level depends heavily on the physical mechanisms and the organization of the computer. It also has many features that are induced by developments in systems software. The specification of a machine level language is an activity which calls for tight cooperation between hardware designers and software designers.

A contribution which is widely regarded as central to the art of programming is the development of key programming mechanisms. Examples of such mechanisms are pushdown stores, "hashing" in memory, subroutines, and macros. While a machine level programming language provides means for expressing in detail the individual steps of a program, the programming mechanisms provide means for organizing and structuring programs at a more global level [24, 45].

Innovations in programming mechanisms have a major impact on the development of hardware and software systems. At times they introduce completely new modes of computing, and frequently they provide ways to increase the efficiency of computer utilization. Experience has shown that the appropriate environment for such innovations is heavy experimental work in computer programming and a climate of exploration with little or no commitment to specific problems.

Programming in machine level language is becoming almost exclusively limited to systems software, where the machine environment is an important factor in the design. Human programming in machine language is rapidly becoming an extinct skill. However, it has strong educational value, as it permits a clear view of fundamental computational processes.

(e) Machine organization schemes; executive and control schemes; design processes. The overall organization of the digital computer has changed little over the years [5, 20]. It captures the essentials of the abstract model of computation—the universal machine—introduced by Turing in the thirties. There have been many variations and refinements of the basic scheme; each has provided solutions to the central pragmatic question of how to implement significant computations at reasonable speed, reliability, and cost. This is an area of vigorous activity in the advanced development and design divisions of the computer industry [2].

There has been a continuous effort to seek new overall organizational schemes for computers. Stimuli for this effort come from new technologies (e.g. integrated electronics induces decentralized designs), from new tasks (e.g. processing visual patterns and problems in meteorology induce parallel computers), and from new models of computer utilization (e.g. time-sharing induces new memory organizations). Work in this area is closely connected with exploratory work on executive and control schemes for managing the operation of computer systems. These efforts are often an integral part of large design activities in operating systems. At present, this general area is one of the most active in the computer field, with computer science departments, industrial laboratories, and various entrepreneurs [15, 26] participating. The emphasis is on innovation, constructive exploration, and promotion.

The introduction of new organizational schemes, together with the growth in complexity of system designs, is underlining the importance of systematizing the processes of computer design. Efforts in description languages, in design methodologies, and in computer-aided design are increasing [37, 42]. Methods of systems analysis and simulation are being developed, and they are being used in many proposed designs [48]. The importance of work on computer-aided design of computers is being widely recognized. In addition to its usefulness for developing specific designs, it contributes basic clarifications to processes of computation, and it provides an excellent vehicle for the study of design processes in general.

Efforts in computer-aided design are stimulating new developments in the area of Graphics [17]. The convenient use of spatial information (charts, diagrams) in man-machine interaction is highly significant for the facilitation of computer usage in a variety of disciplines. The situation is similar to the introduction of high level languages in the field. Work in this area is likely to increase in the coming decade.

(f) Theory of formal languages; automata theory; switching theory. Theoretical activities in these areas are concerned with properties of computer languages, computer mechanisms, and their realizations. Automata theory [19] has a central position, with strong ties to the theory of computation, and to formal languages and switching theory. It is concerned with abstract models of machines and their possible behaviors. Recent work in formal languages has been strongly stimulated by development in structural linguistics (Chomsky [4]) and is currently a very active area of research [21]. Results in this area are relevant to the design of high level languages, translators, and programming mechanisms.

Many of the models studied in automata theory and formal languages have little connection with the real situations that are faced by computer and language designers. As computer science is maturing, there is a growing recognition of the need to bring theoretical work closer to computer design and utilization. For such an enterprise to be successful, it is necessary for potential contributors to have sound knowledge of present theories (and of the relevant background in mathematics and logic), and also to be actively exposed to the body of experience that has accumulated in the field, as well as to the concerns of designers and users.

Work in switching theory is concerned with many of the problems that appear in the logical design of computer subsystems [47]. Activity in this area is slowing down, with many of the contributors moving to problems in automata theory and formal languages. Logical design of computers relies on building blocks (the elementary logic gates and storage elements) that are implemented by electronic circuits. Work on the theory of pulse circuits and on the physical processes that are used for switching and storage (solid state, magnetic, and optical phenomena) is relevant to developments in logical design. Many of these activities take place in electrical engineering departments.

4. Implications on Curriculum Planning

In view of the previous discussion, it is useful to distinguish between two broad types of activities in computer science: (1) activities concerned with problems, methods of solution, and programming; and (2) activities concerned with languages, schemes of processing, and design principles. They correspond roughly to computer applications and to systems.

Professional activities in the computer field include the preparation of specific problems for computer processing and the design and operation of specific software or hardware systems. The effective utilization of computer systems and the planning, design, manufacturing, and maintenance of these systems requires an increasing number of people with training in computer science.

An undergraduate major in computer science must acquire the knowledge and skills needed to hold professional positions in the computer field and to prepare him for further graduate study in computer science. This implies a reasonably broad understanding of the field and the acquisition of working competence in activities of type (1) and/or (2) that are within the state-of-the-art.

A graduate of an M.S. program in computer science must acquire broad knowledge over the entire field, and he must develop considerable competence and professional expertise in at least one of the two types of activities in the field. The M.S. graduate should be able to expand the boundaries of the field by developing new system designs and new applications—mostly within the framework of existing schemes and theories.

A graduate of a Ph.D. program in computer science is expected to contribute substantially to the advancement of the field. He must acquire the knowledge, the skills, and the attitudes that will enable him to do independent research in computer science and to produce new concepts and designs. He is expected to have the breadth of understanding and the problem solving experience that will enable him to explore new and more advanced uses of computers in various domains. Furthermore, he must acquire an overall view of the field and a sufficient exposure to theory and experience, so that he can contribute to the development of an intellectually coherent discipline. This is an important objective for anybody involved in computer science education, faculty and students alike, because of the rapid growth of the field, which causes rapid fragmentation of knowledge and accumulation of large amounts of unstructured detail.

An important part of curriculum planning in any field is to organize and structure the significant knowedge in the field and to map the structured knowledge into a network of interrelated study activities (courses). The structured description of computer science that we have summarized in Figures 1 and 2 provides a good conceptual basis for organizing computer science curricula at different levels. Areas of study and course sequences can be based on different ways of partitioning a structured description and of "tracing" the substructures in each part of the partition.

The partition of the field that we have recently adopted in our planning at Rutgers consists of four main parts that correspond to somewhat overlapping "neighborhoods" in the diagrams of Figures 1 and 2. We identify them as "Hardware Systems," "Software Systems," "Numerical Applications," and "Nonnumerical Applications." Each of these parts determines a *topic of study*. The "Hardware Systems" topic centers on the machine organization schemes activity shown in box (e) in Figure 1 and the theoretical and design activities connected to it. The "Software Systems" topic covers activities in: machine level languages; executive and control schemes; language descriptions and translations; and the activities that are connected to them. The "Numerical Applications" topic covers the top three areas of application in the diagram of Figure 2, and the "Nonnumerical Applications" topic covers the bottom five. In addition, each of the application-oriented topics covers activities that are relevant to it and that are shown in the boxes (a), (b), and (c) of Figure 1: namely, high level languages; representations in computer languages of problems, data, and procedures; and theoretical studies of algorithms.

Specific designs of courses and of overall study plans for students seeking different degrees are made easier and more nearly rational when developed in the proposed framework.

An important part of educational planning in computer science is to continuously examine existing frameworks with a possible view to change them—so that they will better reflect the structure of the changing field. Because of the dynamic character of the field, sudden new developments may occur, and new viewpoints may drastically change our evaluation and mode of presentation of old material. This calls for a flexible and adaptable approach to curriculum planning, both at the global level and at the more detailed levels.

Another implication of the dynamic character of computer science is that students must acquire during their studies the capability to independently adapt to changes in approaches, languages, and systems. This means emphasis on concepts, theories, and general methods, and also on modes of study that promote individual abilities to learn, plan, and work independently.

We are now entering a decade where the interface of computer and society will become universal. As an implication of this it is clear that the responsibility of training leaders in the field becomes great. It is essential for universities to train people with sufficient depth of knowledge in computer science and with a broad enough vision to see the potential of computers for the solution of significant problems in society. Furthermore, the future contributors and leaders in the field must be trained to collaborate with people in other fields in initiating, promoting, and working for change. To achieve this kind of education, we need programs that combine depth in scholarship and adaptability to changes in the field, with sensitivity to the "real world" of problems and emphasis on doing and innovating.

Received July 1970; revised December 1970

References

1. Amarel, S. On the representations of problems and goal-directed procedures for computers. In the *Theoretical Approaches to Non-Numerical Problem Solving*. Banerji and Mesarovic (Eds.), Springer-Verlag, New York, 1970.
2. Blaauw, G.A., et al. The structure of System/360. *IBM Syst. J. 3*, 2 (1964), 119–164.
3. Brooks, F.P. Jr., and Iverson, K.E. *Automatic Data Processing*. Wiley, New York, 1963.
4. Chomsky, N. Formal properties of grammars. In *Handbook of Mathematical Psychology Vol. 2*. R.R. Bush, E.H. Galanter, R.D. Luce (Eds.), Wiley, New York, 1962.
5. Chu, Y. *Digital Computer Design Fundamentals*. McGraw-Hill, New York, 1962.
6. Churchman, C.W. The role of Weltanschuung in problem solving and inquiry. In *Theoretical Approaches to Non-Numerical Problem Solving*. Banerji and Mesarovic (Eds.), Springer-Verlag, New York, 1970.
7. Davis, M. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.
8. Dijkstra, E.W. Notes on structures programming. TH Rep. 70-WSK-03, EWD 249, 2nd ed., Technological U. of Eidhoven, Dep. of Mathematics, Apr. 1970.
9. Feigenbaum, E.A., and Feldman, J. (Eds.) *Computers and Thought*. McGraw-Hill, New York, 1966.
10. Feldman, J., and Gries, D. Translator writing systems. *Comm. ACM 11*, 2 (Feb. 1968), 77–113.
11. Floyd, R. Assigning meanings to programs. Proc. Symposia in Applied Math., Vol. 19, AMS, Providence, R.I., 1967.
12. Galler, B.A. *The Language of Computers*. McGraw-Hill, New York, 1962.
13. Galler, B.A., and Perlis, A. *A View of Programming Languages*. Addison-Wesley, Menlo Park, Calif., 1970.
14. Garvin, P.L., and Spolsky, B. *Computation in Linguistics*. Indiana U. Press, Bloomington, Indiana, 1966.
15. Glaser, E.L., Couleur, J.F., and Oliver, G.A. System design for a computer for time sharing application. Proc. AFIPS 1965 FJCC, Vol. 27, Pt. 1, Spartan Books, New York, pp. 197–202.
16. Gruenberger, F., and Jafray, G. *Problems for Computer Solution*. Wiley, New York, 1965.
17. Gruenberger, F. (Ed.) *Computer Graphics: Utility/Production Art*. Thompson Books, Washington, D.C., 1967.
18. Hamming, R.W. One man's view of computer science. 1968 ACM Turing Lecture; *J. ACM*, *16*, 1, (Jan. 1969), 3–12.
19. Harrison, M. *Introduction to Switching and Automata Theory*. McGraw-Hill, New York, 1965.
20. Hellerman, H. *Digital Computer System Principles*. McGraw-Hill, New York, 1967.
21. Hopcroft, J.E., and Ullman, J. *Formal Languages and Their Relations to Automata*. Addison-Wesley, Menlo Park, Calif., 1969.
22. Isaacson, E., and Keller, H.B. *Analysis of Numerical Methods*. Wiley, New York, 1966.
23. Kilmer, W.L., and Blum, J. Some mechanisms for a theory of the reticular formation. Final Rep., AFOSR 67-0928, AD651207, Feb. 1967.
24. Knuth, D.E. *The Art of Computer Programming Fundamental Algorithms Vol. I*. Addison Wesley, Menlo Park, Calif., 1968.
25. Lavi, A., and Voge, E. (Eds.) *Recent Advances in Optimization Techniques*. Wiley, New York, 1966.
26. Lampson, B.W. Scheduling philosophy for multiprocessing systems. *Comm. ACM 11*, 5 (May 1968), 347–360.
27. Lewis, P.M., and Coates, C.L. *Threshold Logic*. Wiley, New York, 1967.
28. The mathematical sciences: A report. Pub. 1681, Nat. Acad. of Sci., Washington, D.C., 1968, p. 94.
29. Mendel, J., and Fu, K. *Adaptive Learning and Pattern Recognition Systems*. Academic Press, New York, 1970.
30. McCarthy, J., and Hayes, P. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, Michie and Meltzer (Eds.), Edinburgh U. Press, Edinburgh, 1969.
31. McCarthy, J. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*. Braffort and Hershberg (Eds.), North Holland Pub. Co., Amsterdam, 1963.
32. Minsky, M. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N.J., 1967.
33. Minsky, M., and Papert, S. *Perceptrons*. MIT Press, Cambridge, Mass., 1969.
34. Newell, A. Heuristic programming: Ill structured problems. In *Progress in Operations Research III*. J.S. Aronofsky (Ed.), Wiley, New York, 1969.
35. Newell, A., and Simon, H.A. Programs as theories of higher mental processes. In *Computers in Biomedical Research, Vol. 2*. Stacy and Waxman (Eds.), Academic Press, New York, 1965.
36. Nilsson, N. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York (in press, spring 1971).
37. Parnas, D.L., and Darringer, J.A., SODAS and a methodology for system design. Proc. AFIPS 1967 FJCC, Vol. 31, AFIPS Press, Montvale, N.J., pp. 449–474.
38. Salton, G. *Automatic Information Organization and Retrieval*. McGraw-Hill, New York, 1968.
39. Sammet, J.E. *Programming Languages: History and Fundamentals*. Prentice-Hall, Englewood Cliffs, N.J., 1969.
40. Sammet, J.E. Annotated descriptor-based bibliography on the use of computers for non-numerical mathematics. *Comput. Rev. 7*, 4 (July–Aug. 1966).
41. Simon, H. *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., 1969.
42. Srinivasan, C.V. CDL1, a computer description language, Pt. 1: The nature of the description language and organization of descriptions. Proc. 3rd Ann. Princeton Conf. on Inf. and System Sci., Princeton U., Mar. 1969.
43. Tocher, K.D. *The Art of Simulation*. Van Nostrand, Princeton, N.J., 1963.
44. Van Wijngaarden (Ed.) Report on the algorithmic language Algol-68 *Numerchische Mathematic 14*, 1969.
45. Wegner, P. *Programming Languages, Information Structures, and Machine Organization*. McGraw-Hill, New York, 1968.
46. Wirth, N. PL360, a programming language for the 360 computers. *J. ACM 15*, 1 (Jan. 1968), 37–74.
47. Wood, P.E. *Switching Theory*. McGraw-Hill, New York, 1968.
48. Zurcher, F.W., and Randell, B. Iterative multilevel modeling: A methodology for computer system design. Proc. IFIP Cong. 1968, Vol. 2, North Holland Pub. Co., Amsterdam, pp. 867–871.