COMPUTER SCIENCE: THE DISCIPLINE

Peter J. Denning

August 1997
Revised July 1999

This article will appear in the 2000 Edition of *Encyclopedia of Computer Science* (A. Ralston and D. Hemmendinger, Eds).

The computing profession is the people and institutions that have been created to take care of other people's concerns in information processing and coordination through worldwide communication systems. The profession contains various specialties such as computer science, computer engineering, software engineering, information systems, domain-specific applications, and computer systems. The discipline of computer science is the body of knowledge and practices used by computing professionals in their work. This article, with many cross-references to other articles in this encyclopedia, discusses these aspects of the profession and the relations among them.

The discipline of computer science was born in the early 1940s with the confluence of algorithm theory, mathematical logic, and the invention of the stored-program electronic computer. Examples are the works of Alan Turing and Kurt Godel in the 1930s about algorithms and their realizations as machines or rule-systems, the algorithms created by Ada Lovelace sixty years earlier, the analog computers built by Vannevar Bush in the 1920s, and the electronic computers built by Howard Aiken and Konrad Zuse in the 1930s. The writings of John von Neumann demonstrate considerable intellectual depth to the emerging discipline by the late 1940s. By the early 1960s, there was a sufficient body of knowledge to merit the first academic departments and degree programs. This discipline is also called computer science and engineering, computing, and informatics.

The body of knowledge of computing is frequently described as the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, *What can be (efficiently) automated?*

This common characterization is too austere. It only hints at the full richness of the discipline. It does not call attention to the connections between computing knowledge and the concerns of people to whom this knowledge contributes, notably the universal concerns for reliability, dependability, robustness, integrity, security, and modifiability of computer systems. It hides the social and historical context of the field and the values of the people who practice in it. The following discussion of the discipline calls attention to these important larger questions.

The Domain of Computer Science

Even though computer science addresses both human-made and natural information processes, the main effort in the discipline has been directed toward human-made processes, especially information processing systems and machines. Much of the work of the field until the mid 1980s concerned computers as number-crunchers, symbol-manipulators, and data processors; but the personal computer and the Internet have since enlarged the focus to include coordination and communication. Much of the body of knowledge of computing concerns the digital computer and the phenomena surrounding it --- the structure and operation of computer systems, principles underlying computer system design and programming, effective methods for using computers for information processing tasks, and theoretical characterizations of their properties and limitations. The field is grappling with new questions arising from interactions with other fields, where computers are tools but not the objects of study, and where other considerations such as transparency, usability, dependability, reliability, and safety are paramount.

Computing is also contributing to other fields by showing them how to model their processes as information processes. Several have been prominent. Biologists now view DNA as an encoding of information needed to generate a unique organism; they seek algorithms that can construct complete genome sequences from fragments scattered across many databases. Psychologists and cognitive scientists, who have collaborated for many years with computer scientists on models of cognition, have been joined by neuroscientists who seek to model neural systems and use the models to explain cognitive behavior of organisms. Physicists have discovered new materials that have been used for ever-smaller chips and ever-faster communication media. Other disciplines are beginning to contribute ideas for the construction of new machines, such as silicon chips that simulate body parts like eyes and ears, biological memories, DNA chemical solutions that compute combinatorial problems, or quantum processes for superparallel computation and cryptography.

Standard Concerns of the Field

The digital computer plays the central role in the field because it is a universal computing machine: with enough memory, a digital computer is capable of simulating any information processing system, provided the task can be specified as an unambiguous set of instructions. If such a specification is possible, the task can be represented as a program that can be stored in the memory of a computer. Thus, one machine is capable of exploring and studying an enormous variety of concepts, schemes, simulations, and techniques of information processing.

Every practitioner of the discipline must be skilled in four basic areas: algorithmic thinking, representation, programming, and design. Algorithmic thinking is an interpretation of the world in which a person understands and formulates actions in terms of step-by-step procedures that give unambiguous results when carried out by anyone (or by a suitable machine). It resembles standard scientific thinking, which seeks to invent standard ways of observing that allow anyone to see and reproduce physical effects. Algorithmic thinking emphasizes the standard procedure and scientific thinking the standard observer.

Representation addresses the way in which data are stored so that the questions one will ask about them can be answered efficiently. For example, the standard phone book is organized for quick answer of the question, "What is the phone number assigned to person N?" When stored in a computer database, the machine goes quickly to the alphabetic part of the list containing the name N. This is not an efficient organization for the question, "To whom is the phone number N assigned?" because the standard book would have to be searched entry by entry until the given phone number is found. A data organization more suited to the second question is the inverted phone book, in which the phone numbers are listed in their numeric order, accompanied by the names of their owners. The skill of representation goes beyond knowing how to organize data for efficient retrieval or processing. It deals with inventing ways of encoding phenomena to allow algorithmic processing. Examples include representing a mathematical expression so that it can be differentiated, representing a document so that "What you see [on the screen] is what you get [on the printer]"; representing handwritten zip codes for automatic recognition and sorting by the Postal Service; representing encoded speech so that one can talk to a computer or so that the computer can talk; representing an engine part so that it can be shown on the graphics screen and can also be manufactured automatically.

Programming enables people to take algorithmic thinking and representations and embody them in software that will cause a machine to

perform in a prescribed way. This skill includes working knowledge of different programming languages (each having its own strengths and limitations), program development tools (which aid testing, debugging, modularity, and compatibility), and operating systems (which control the internal operations of computers).

Design connects the other three skills to the concerns of people, though the medium of systems that serve them. Design includes many practical considerations such as engineering tradeoffs, integrating available components, meeting time and cost constraints, and meeting safety and reliability requirements.

Even though everyone in the discipline is expected to know these skills, it is a mistake to equate computer science with any one of them, e.g., programming. As will be shown shortly, there are many aspects of the discipline that do not involve programming even though they involve algorithmic thinking, representation, and design.

Principal Subdivisions of the Field

The subject matter of computing can be broadly divided into two parts. The first studies information processing tasks and their related data representations. The second studies structures, mechanisms, and schemes for processing information. Within the field, these two parts are called applications and systems, respectively. A major goal of computing education is to elucidate the relationships between applications and computer systems.

Computer applications can be subdivided into numerical and nonnumerical categories. Numerical applications are those in which mathematical models and numerical data are dominant; they are supported by numerical analysis, optimization, simulation, mathematical libraries, computational geometry, and computational science. Nonnumerical applications are those in which problems and information are represented as symbols and rules; they are supported by artificial intelligence, multimedia systems, language processors, graphics, mathematical expression systems, database systems, information retrieval systems, and combinatorial processes.

Computer systems can be subdivided into software systems and hardware systems. Software systems are concerned with machine-level representations of programs and data, schemes for controlling program execution, compilers, programming environments, network communications and management, and operating systems. Hardware systems are concerned with logical design, machine organization, processors, memory, and devices in various

technologies such as VLSI, silicon, and GaAs. Computer architecture and computer engineering are concerned with both software and hardware.

These categories do not define clean lines of division. Most application areas are also concerned with related systems problems such as languages, operating systems, and networks. Most systems areas are also concerned with task environments, practices of the application area, and modes of human interaction.

Relationships with Other Disciplines

Computer science has, by tradition, been more closely related to mathematics than physics, chemistry and biology. This is because mathematical logic, the theorems of Turing and Godel, Boolean algebra for circuit design, and algorithms for solving equations and other classes of problems in mathematics played strong roles in the early development of the field. Conversely, computer science has strongly influenced mathematics, many branches of which have become concerned with demonstrating algorithms for constructing or identifying a mathematical structure or carrying out a function. In some cases, computers have been essential to mathematics; for example, the solution of the four-color theorem relied on a program that searched a large finite number of cases for counterexamples. For these reasons, some observers like to say that computing is a mathematical science.

The bond between engineering and computer science is much stronger than between many natural science disciplines and their engineering counterparts -- for example, chemical engineering and chemistry, aircraft design and fluid dynamics, pharmacy and biology, and materials engineering and physics. This is because computer science has a strong heritage in electrical engineering and because many algorithmic methods were designed originally to solve engineering problems. Examples include electronic circuits, telecommunications, engineering graphics, engineering design, systems engineering, fabrication, and manufacturing. Conversely, computers have become indispensable in many engineering disciplines -- for example, circuit simulators, finite-element simulators, flow-field simulators, graphics, CAD and CAM systems, computer controlled tools, and flexible manufacturing systems. For these reasons, some observers like to say that computing is an engineering science.

A new bond is forming between the physical sciences and computer science. Leaders of physics, chemistry, biology, geology, seismology, astronomy, oceanography, and meteorology have brought to prominence certain very hard, “grand challenge” problems that demand massive high-speed computations, performed on new generations of massively parallel

computers with new kinds of algorithms. These problems include crystalline structure, quantum electrodynamics, calculation of chemical properties of materials from the Schroedinger equation, simulation of aircraft in flight, exploration of space, global climate modelling, oil exploration, models of the universe (cosmology), long range weather forecasting, earthquake prediction, turbulent fluid flow, and human genome sequencing. Many leaders of science now say that computation has emerged as a third paradigm of science, joining theory and experimentation. For these reasons, some observers identify computing with computational science.

Who's right? All are, demonstrating the richness of the discipline and its heritage in older sciences and engineering. In addition to the influences of mathematics, engineering, and science woven into the discipline itself, computing interacts closely with many other disciplines. Here are some prominent examples:

- Library science is concerned with archiving texts and organizing storage and retrieval systems to give efficient access to texts. As digital library systems are built and attached to the Internet, libraries will change from storage places for books to electronic data centers, and will grant access well beyond their local communities. Libraries have a special concern with the problem of migrating data from older storage media onto newer ones.

- Management science is concerned with using computer models for planning and forecasting economic conditions for business. It is also concerned with storing business records in databases and generating reports on the state of the business and on customer preferences from these records.

- Economics is concerned with using computer models to forecast economic conditions and to evaluate the possible effects of macro-economic policies.

- Medicine and Biology have used computer models and algorithms in ingenious ways to diagnose and treat diseases. Modern imaging methods such as Magnetic Resonance Scans, Coronary Scans, and Tomography have drawn heavily on computer science. Medical researchers use computer models to assist them in tracking mutations of viruses and in narrowing the scope of experiments to the cases most likely to resolve the research question. The Human Genome project has used large distributed databases and new kinds of string-matching algorithms to aggregate the tens of thousands of DNA sequencing experiments.

- Forensics uses computer models and large databases to identify evidence and discover whether other forensic data matches current evidence.

- Psychology, Cognitive, and Behavioral sciences are concerned with understanding human thought and emotions. They use computer models to gain insight into the operation of human brains and nervous systems and to design effective interventions into human problems.

- Linguistics is concerned with using computers to recognize speech, translate between languages, and to understand the role of language in human affairs.

- Philosophy is concerned with the way people acquire knowledge, create social realities, and act morally and ethically. Philosophers have contributed much to the debates on whether machines can think or whether formal models are sufficient for dependable software systems. The subdiscipline of speech act theory has contributed much to our understanding of how people carry out work in organizations and has helped give birth to the workflow industry. Recently, the technologies of "virtual realities" have rekindled debates on the nature of reality and the worlds in which people live.

- Humanities have begun to use computer extensively to correlate and search through historical artifacts that can be represented digitally. One of the more colorful examples is the use of computers to determine authorship of historical texts, such as Shakespeare's plays.

This list is hardly exhaustive. The number of contacts between computing and other disciplines grows rapidly each year. Some of the most innovative work is being done by people who know another discipline and computing at once.

Processes

At the beginning of the previous section, we noted that mathematics, science, and engineering have special historical relationships with computing. These roots show up as three major paradigms or processes within the field:

- THEORY: Building conceptual frameworks and notations for understanding relationships among objects in a domain and the logical consequences of axioms and laws.

- EXPERIMENTATION: Exploring models of systems and architectures within given application domains and testing whether those models can

predict new behaviors accurately. (This paradigm is sometimes called <I>abstraction</I> by computer scientists.)

· DESIGN: Constructing computer systems that support work in given organizations or application domains.

These three paradigms constantly interact in the work of computer scientists; indeed, the interaction is part of the vigor of the field. Many controversies in the field are associated with someone in one paradigm criticizing the work of someone in another without being aware of the difference.

In areas of rapidly developing technology, such as databases, human interfaces, and Web-based systems, theoreticians aim mainly at bringing order into a rapidly accumulating mass of experience through broad conceptual frameworks, taxonomies, and analytic methods. In mature areas such as computational complexity, algorithms, data structures, automata, formal languages, switching theory, graph theory, combinatorics, and formal languages, theoreticians focus on deeper, comprehensive analyses of phenomena for which formal models exist. With a few notable exceptions including logic design, graphics, algorithm analysis, and compilers, theory has had limited impact on the complex problems of practical systems and applications.

Experimenters construct models of phenomena or of possible systems; the models generally suppress detail and enable fast predictions. Examples are measurement of programs and systems, validation of hypotheses, prototyping to extend abstractions to practice, logic simulation, simulations of systems and of physical processes, testing of protocols, system performance analysis, and comparisons of different architectures. Experimental computer science relies heavily on laboratories. It often stimulates new developments in computer design and use. More attention is being paid to experimental computer science because human intuition often does not work well with complex systems.

Designers are concerned with building systems that meet clear specifications and satisfy their customers. They boast many significant accomplishments such as program development systems, simulators, microchip design systems, VLSI, CAD, CAM, graphics, databases, and supercomputers. The most successful designs have occurred with hardware and self-contained software packages -- systems for which precise functional specifications can be given at the start. The least successful designs have been large software systems, many of which are unreliable, undependable, unsafe, too costly, too difficult to change, and too complex to understand. Many designers are turning to other domains including organizational analysis, workflow, anthropology, and ethnography to assist them in understanding how a system will interact with the practices of the people using them.

In addition to the three processes and the specialists who practice them, the discipline of computing has a number of broad concerns that touch all the subfields. The main ones are parallel and distributed computation, performance analysis, reliability, safety, security, and ethics.

Subareas of the Field

Computer science can be divided into a number of coherent subareas, each with substantial theoretical, experimental, and design issues, and each with its own version of the shared concerns. Significant industries and institutions have been established in each of these areas. The chart below depicts the discipline as a matrix with 11 subareas as rows and the 3 processes as columns. Each of the boxes can be filled in with detailed descriptions of that category of the subarea's activities and accomplishments. The boundaries between areas and processes are often fuzzy; it is sometimes a matter of personal judgment where certain items go. Additional columns could be added to represent the shared concerns, and their boxes filled in likewise.

The discussion following is an overview of the content of the boxes of the matrix, with just enough depth to reveal the language and vocabulary of computer science. Much more information about these areas can be found in the individual articles of this volume and in the Handbook of Computer Science and Engineering (Tucker 1996). The last area, bioinformatics, is an emerging area.

	Theory	**Abstraction**	**Design**
1 Algorithms & Data Structures			
2 Programming Languages			
3 Architecture			
4 Operating Systems and Networks			
5 Software Engineering			
6 Databases & Information Retrieval			
7 Artificial Intelligence & Robotics			
8 Graphics			
9 Human Computer Interaction			
10 Computational Science			
11 Organizational Informatics			
12 Bioinformatics			

1 Algorithms and Data Structures

The theory of algorithms encompasses computability theory, computational complexity theory, concurrency theory, probabilistic algorithm theory, database theory, randomized algorithms, pattern-matching algorithms, graph and network algorithms, algebraic algorithms, combinatorial optimization, and cryptography. It is supported by discrete mathematics (graph theory, recursive functions, recurrence relations, combinatorics), calculus, induction, predicate logic, temporal logic (a calculus of time dependent events), semantics, probability, and statistics.

Experimentation has been found very useful with complex algorithms and heuristics for which no tractable theoretical analysis is known. Algorithms can be evaluated by applying them to suites of test cases and analyzing their performance. Testing has yielded valuable characterizations of certain methods such as divide-and-conquer, greedy algorithms, dynamic programming, finite state machine interpreters, stack machine interpreters, heuristic searches, and randomized algorithms. Testing has yielded significant insights into the performance of parallel and distributed algorithms.

Many useful, practical algorithms have been designed and placed in program libraries -- for example, mathematical software, searching, sorting, random-number generation, textual pattern matching, hashing, graphs, trees, communication network protocols, distributed-data updates, semaphores, deadlock detectors, synchronizers, storage managers, lists, tables, and paging algorithms. Many theoretical results have been translated into useful and practical systems, such as the RSA public key cryptosystem, production-quality compilers, and VLSI circuit layout.

2 Programming Languages

This area deals with notations for virtual machines that execute algorithms and with notations for algorithms and data; the sets of strings of symbols that are generated by such notations are called languages. It also deals with efficient translations from high-level languages into machine codes. Fundamental questions include: What are possible organizations of the virtual machine presented by the language (data types, operations, control structures, mechanisms for introducing new types and operations)? How are these abstractions implemented on computers? What notation (syntax) can be used effectively and efficiently to specify what the computer should do? How are functions (semantics) associated with language notations? How can machines translate between languages?

The theory of programming languages studies models of machines that generate and translate languages and of grammars for expressing valid strings

in the languages. Examples include models of formal languages and automata, Turing machines, Post systems, lambda-calculus, pi-calculus, and propositional logic. The theory deals with semantics, the study of the relationships between strings of the language and states of the underlying virtual machines. It deals with types, which are classes of objects. Related mathematics is predicate logic, temporal logic, modern algebra, and mathematical induction.

The modelers have developed classifications of languages based on their syntactic and semantic models, for example, static typing, dynamic typing, functional, procedural, object-oriented, logic specification, message-passing, and dataflow. They have developed classifications by application, for example, business data processing, simulation, list processing, and graphics. They have developed classifications by functional structure, for example, procedure hierarchies, functional composition, abstract data types, and communicating sequential processes. They have developed abstract implementation models for each major type of language including imperative, object-oriented, logic and constraint, concurrent, and distributed.

Programming language designers have developed many practical languages including procedural languages (Cobol, Fortran, Algol, Pascal, Ada, and C), object-oriented languages (Clu, Smalltalk, C++, Eiffel, Java), functional languages (Lisp, ML, Haskell), dataflow languages (Sisal, Val, Id Nouveau), logic (Prolog), string (Snobol, Icon), and concurrency (Concurrent Pascal, Occam, SR, Modula-3).

They have implemented run-time models, static and dynamic execution models, type checking, storage and register allocation, compilers, cross compilers, interpreters, analyzers that find parallelism in programs, and programming environments that aid users with tools for efficient syntactic and semantic error checking, profiling, debugging, and tracing. A crowning achievement has been programs that take the description of a language and produce automatically a compiler that will translate programs in that language into machine code (examples include YACC and LEX in Unix environments); very efficient compilers have been built this way. Programming languages are used widely in application domains to create tables, graphs, chemical formulas, spreadsheets, equations, input and output, and data queries; in each case, the designer creates a mini-language and a parser.

3 Architecture

This area deals with methods of organizing hardware (and associated software) into efficient, reliable systems. The fundamental questions include: What are good methods of implementing processors, memory, and

communication in a machine? How does one design and control large computational systems and convincingly demonstrate that they work as intended despite errors and failures? What types of architectures can efficiently incorporate many processing elements that can work concurrently on a computation? How does one measure performance? Can hardware devices mimic selected human sensors such as eyes and ears?

The theory of architecture includes: digital logic, Boolean algebra, coding theory, and finite-state machine theory. Supporting mathematics include statistics, probability, queueing theory, reliability theory, discrete mathematics, number theory, and arithmetic in different number systems.

Computer architects are avid experimenters. Their favorite models include finite state machines, general methods of synthesizing systems from basic components, models of circuits and finite state machines for computing arithmetic functions over finite fields, models for data path and control structures, optimizing instruction sets for various models and workloads, hardware reliability, space, time, and organizational tradeoffs in the design of VLSI devices, organization of machines for various computational models, and identification of "levels of abstraction" at which the design can be viewed -- e.g., configuration, program, instruction set, registers, and gates. Architects frequently use simulators to assess design tradeoffs and determine the best ratios of memory, processing power, and bandwidth for a device. They have well-developed discourses for buses (inter-device data channels), memory systems, computer arithmetic, input and output, and parallel machines.

Computer architecture is replete with successful designs. These include arithmetic function units, cache, the so-called von Neumann machine, RISCs (Reduced Instruction Set Computers), CISCs (Complex Instruction Set Computers), efficient methods of storing and recording information and of detecting and correcting errors; error recovery, computer aided design (CAD) systems and logic simulations for the design of VLSI circuits, reduction programs for layout and fault diagnosis, silicon compilers (compilers that produce instructions for manufacturing a silicon chip). They also include major systems such as dataflow, tree, Lisp, hypercube, vector, and multiprocessors; and supercomputers, such as the Cray, Cyber, and IBM machines. Architects have collaborated with other scientists to design prototypes of devices that can imitate human senses.

4 Operating Systems and Networks

This area deals with control mechanisms that allow multiple resources to be efficiently coordinated in computations distributed over many computer systems connected by local and wide-area networks. Fundamental questions include: At each level of temporal granularity (e.g. microsecond, minute,

hour, or day) in the operation of a computer system, what are the visible objects and permissible operations on them? For each class of resource (objects visible at some level), what is a minimal set of operations that permit their effective use? How can interfaces be organized so that users deal only with abstract versions of resources and not with physical details of hardware? What are effective control strategies for job scheduling, memory management, communications, access to software resources, communication among concurrent tasks, reliability, and security? What are the principles by which systems can be extended in function by repeated application of a small number of construction rules? How should distributed computations be organized, with the details of network protocols, host locations, bandwidths, and resource naming being mostly invisible? How can a distributed operating system be a program preparation and execution environment?

Major elements of theory in operating systems include: concurrency theory (synchronization, determinacy, and deadlocks); scheduling theory; program behavior and memory management theory; network flow theory; performance modeling and analysis. Supporting mathematics include bin packing, probability, queueing theory, queueing networks, communication and information theory, temporal logic, and cryptography.

Like architects, operating system designers are avid modelers. Their major models include: abstraction and information-hiding principles; binding of user-defined objects to internal computational structures; process and thread management; memory management; job scheduling; secondary storage and file management; performance analysis; distributed computation; remote procedure calls; real-time systems; secure computing; and networking, including layered protocols, Internet protocols, naming, remote resource usage, help services, and local network routing protocols such as token-passing and shared buses.

Operating systems and networking has always been, first and foremost, a field of design. This field has yielded efficient standard methods including time sharing systems, automatic storage allocators, multilevel schedulers, memory managers, hierarchical file systems. It has yielded well-known operating systems such as Unix, Multics, Mach, VMS, MacOS, OS/2, MS-DOS, and Windows NT. It has produced standard utilities including editors, document formatters, compilers, linkers, and device drivers. It has produced standard approaches to files and file systems. It has produced queueing network modeling and simulation packages for evaluating performance of real systems; network architectures such as Ethernet, FDDI, token ring nets, SNA, and DECNET. It has produced protocol techniques embodied in the US Department of Defense protocol suite (TCP/IP), virtual circuit protocols, Internet, real time conferencing, and X.25. It has devoted considerable attention to security and privacy issues in the Internet.

5 Software Engineering

This area deals with the design of programs and large software systems that meet specifications and are safe, secure, reliable, and dependable. Fundamental questions include: What are the principles behind the development of programs and programming systems? How does one make a map of the recurrent actions people take in a domain and use the map to specify a system of hardware and software components to support those actions? How does one prove that a program or system meets its specifications? How does one develop specifications that do not omit important cases and can be analyzed for safety? By what processes do software systems evolve through different generations? By what processes can software be designed for understandability and modifiability? What methods reduce complexity in designing very large software systems?

Three kinds of theory are used for software engineering: program verification and proof (which treats forms of proofs and efficient algorithms for constructing them), temporal logic (which is predicate calculus extended to allow statements about time-ordered events), and reliability theory (which relates the overall failure probability of a system to the failure probabilities of its components over time). Supporting mathematics include predicate calculus and axiomatic semantics. Software engineering also draws on theory from cognitive psychology.

Models and measurements play important roles in software engineering. There are nine major categories. (1) Specification of the input-output functions of a system: predicate transformers, programming calculi, abstract data types, object-oriented notations, and Floyd-Hoare axiomatic notations. (2) The process by which a programmer constructs software: stepwise refinement, modular design, separate compilation, information-hiding, dataflow, software lifecycle models, layers of abstraction. (3) Processes to develop software systems: specification-driven, evolutionary, iterative, formal, and cleanroom. (4) Processes to assist programmers in avoiding or removing bugs in their programs: syntax-directed text editors, stepwise program execution tracers, programming environments, and software tools. (5) Methods to improve the reliability of programs: software fault tolerance, N-version programming, multiple-way redundancy, checkpointing, recovery, information flow security, testing, and quality assurance. (6) Measurement and evaluation of programs. (7) Matching software systems with machine architectures (the more specialized high-performance computers are not general-purpose). (8) Organizational strategies and project management. (9) Software tools and environments.

Like operating systems, software engineering is primarily a design specialty. Many of the models noted above have been used in practice under particular

designations. Examples of specification languages include PSL2 and IMA JO. Software projects use version control systems to track versions of the modules of the emerging system; examples are RCS and SCCS. Many syntax-directed editors, line editors, screen editors, and programming environments have been implemented; examples are Turbo C and Turbo Pascal. Methodologies for organizing the software development process go under generic names like HDM or the names of their inventors (e.g., Dijkstra, Jackson, Mills, Yourdon, Weinberg). These process methodologies incorporate procedures for testing, quality assurance, and overall project management (e.g., walk-through, hand simulation, interface checking, program path enumerations for test sets, and event tracing). The US Department of Defense has promulgated additional criteria and testing methods for secure computing. Many software tools have been built to assist with program development, measurement, profiling, text formatting, and debugging. A significant number of designers are concerned with the user interface; especially of systems on which human lives depend, they seek to organize the user interface to minimize the possibility of human misinterpretation, especially in times of stress. The Internet's growth to over 20 million computers worldwide by 1998 has generated a new specialty in designing computations with component processes on individual computers around the world; it goes under various names such as programming-in-the-large, distributed program composition, and megaprogramming.

6 Database and Information Retrieval Systems

This area deals with the organization of large sets of persistent, shared data for efficient query and update. The term database is used for a collection of records that can be updated and queried in various ways. The term retrieval system is used for a collection of documents that will be searched and correlated; updates and modifications of documents are infrequent in a retrieval system. Fundamental questions include: What models are useful for representing data elements and their relationships? How can basic operations such as store, locate, match, and retrieve be combined into effective transactions? How can the user interact effectively with these transactions? How can high-level queries be translated into high-performance programs? What machine architectures lead to efficient retrieval and update? How can data be protected against unauthorized access, disclosure, or destruction? How can large databases be protected from inconsistencies due to simultaneous update? How can protection and performance be achieved when the data are distributed among many machines? How can text be indexed and classified for efficient retrieval?

A variety of theories have been devised and used to study and design database and information retrieval systems. These include relational algebra and relational calculus, concurrency theory, serializable transactions, deadlock

prevention, synchronized updates, statistical inference, rule-based inference, sorting, searching, indexing, performance analysis, and cryptography as it relates to ensuring privacy of information and authentication of persons who stored it or attempt to retrieve it.

Models and associated measurements have been used in at least nine ways. (1) Data models for the logical structure of data and relations among data elements: object-based, record-based, and object-relational. (2) Storing files for fast retrieval, notably indexes, trees, inversions, and associative stores. (3) Access methods. (4) Query optimization. (5) Concurrency control and recovery. (6) Integrity (consistency) of a database under repeated updates, including concurrent updates of multiple copies. (7) Database security and privacy, including protection from unauthorized disclosure or alteration of data and minimizing statistical inference. (8) Virtual machines associated with query languages (e.g., text, spatial data, pictures, images, rule-sets). (9) Hypertext and multimedia integration of different kinds of data (text, video, graphics, voice).

A rich set of practical design techniques exists for the database or retrieval system designer. They include general approaches to relational, hierarchical, network, distributed databases, and retrieval systems. They are used in commercial database systems such as Ingres, System R, dBase, Sybase, and DB-2, and in commercial retrieval systems such as Lexis, Osiris, and Medline, and in commercial hypertext systems such as NLS, NoteCards, HyperCard, SuperCard, Intermedia, and Xanadu. Many techniques have been created for secure database systems in which data are marked by classification level and compartment and users cannot see data inconsistent with their own classification levels and compartments. There are standard methods of archiving data sets onto long-term media such as tape and optical disk, along with methods of migrating large data sets from older to newer media. Large-capacity media such as CD/ROM have become sufficiently inexpensive that many commercial information products such as literature reviews, selected papers and book extracts, and software magazines are available.

7 Artificial Intelligence and Robotics

This area deals with the modeling of animal and human cognition, with the ultimate intention of building machine components that mimic or augment them. The behaviors of interest include recognizing sensory signals, sounds, images, and patterns; learning; reasoning; problem-solving; planning; and understanding language. Fundamental questions include: What are basic models of cognition and how might machines simulate them? How can knowledge of the world be represented and organized to allow machines to act reasonably? To what extent is intelligence described by search, heuristics, rule evaluation, inference, deduction, association, and pattern computation?

What limits constrain machines that use these methods? What is the relation between human intelligence and machine intelligence? How are sensory and motor data encoded, clustered, and associated? How can machines be organized to acquire new capabilities for action (learning), make discoveries, and function well despite incomplete, ambiguous, or erroneous data? How might machines understand natural languages, which are replete with ambiguities, paraphrases, ellipses, allusions, context, unspoken assumptions, and listener-dependent interpretations? How can robots see, hear, speak, plan, and act?

Nine major branches of theory have been developed for artificial intelligence. (1) Logic systems for mechanical reasoning such as first-order logic, fuzzy logic, temporal logic, non-monotonic logic, probabilistic logic, deduction, and induction. (2) Formal models for representing and translating knowledge including objects, grammars, rules, functions, frames, and semantic networks. (3) Methods for searching the very large spaces that arise when enumerating solutions to problems; these include branch-and-bound, alpha-beta, tree pruning, and genetic algorithms. (4) Theories of learning including inference, deduction, analogy, abduction, generalization, specialization, abstraction, concretion, determination, and mutual dependency. (5) Neural networks deal with neural interconnection structures, computing responses to stimuli, storing and retrieving patterns, and forming classifications and abstractions. (6) Computer vision. (7) Speech recognition and understanding. (8) Natural language translation. (9) Robot systems. All branches of the theory draw heavily on the related disciplines of structural mechanics, graph theory, formal languages, linguistics, logic, probability, philosophy, and psychology.

Models and measurements have been used extensively in various subdomains of intelligent systems and learning machines. (1) Knowledge representation models include rules, frames, logic, semantic networks, neural networks, deduction, forward and backward inference, inheritance, instantiation, resolution, spreading activation, backward error propagation. (2) Problem-solving models include case-based reasoning, qualitative reasoning, constraint-based and opportunistic reasoning, distributed and cooperative reasoning, and nonlinear planning. (3) Heuristics for searching large spaces of alternatives are at the heart of efficient machines for checkers, chess, and other games of strategy. Heuristic searching has spawned a new field called Evolutionary Computation, which are methods inspired by the biological principle of the evolution of a population of candidates in which the best ones become predominant. The most well known among these are genetic algorithms. (4) Learning models have improved the ability of machines to solve problems; these include knowledge acquisition from data or experts, learning rules from examples, revising theories, discovering patterns and quantitative laws in data sets, data classification and clustering, and multi-strategy learning models. (5) Language understanding models have helped to represent syntactic and semantic forms, find answers to

questions, and translate between languages. (6) Speech models are used to produce speech (with good results) and recognize speech (still relatively primitive). (6) Vision models offer algorithms for finding and recognizing objects in visual fields. (7) Neural network models have been tested extensively to evaluate their ability to store patterns, remove noise from patterns, retrieve patterns, generalize patterns, and to store large numbers of patterns without loss or interference. (8) Models of human memory store large patterns and form associations between them. (9) Knowledge robots ("knowbots") have been proposed for the Internet to make discoveries, classify objects, or deduce description rules for large data sets and time-series data.

Artificial intelligence has fostered many implementations and its own design principles. (1) Logic programming has been realized in languages, notably Prolog, based on efficient theorem proving, rule resolution, and rule evaluation. (2) Expert systems, which use an inference engine to process rules stored in a knowledge base to deduce and propose actions, have been successful in a number of well-focused, narrow domains such as medical diagnosis, planning, machine repair, system configuration, and financial forecasting. (3) Knowledge engineering environments can be instantiated into expert systems when their knowledge bases are loaded with rules and facts for the domain in which they will be used. (4) Natural-language problem-solving systems (e.g., Margie and SHRDLU) have been successful in limited cases. (5) Games of strategy, notably checkers and chess, are played by machines at world-champion levels. (6) Neural networks have been used for pattern recognition, speech recognition, vision recognition, simulation of human long-term memory, and evaluating student competence. (7) Fuzzy logic has been implemented on chips that make control systems in common appliances such as air conditioners and washing machines. (8) Speech synthesizers are widely available. (9) Speech recognizers are becoming common and can already do well with continuous speech by an individual on whose voice the system was trained. (10) Robots are standard in assembly lines, Mars rovers, and even routine tasks such as household cleaning or lab maintenance. Robot insects have been built as models of ambulatory machines that can perform simple tasks such as cleaning and exploring. (11) Genetic algorithms have been used in numerous applications.

8 Graphics

This area is concerned with processes for representing physical and conceptual objects and their motions visually on a 2D computer screen or in a 3D hologram. Fundamental questions include: What are efficient methods of representing objects and automatically creating pictures for viewing? For projecting motions of complex objects onto the viewing screen in real time? For displaying data sets to aid human comprehension? For virtual realities --

i.e., simulations of real situations that are difficult to distinguish from the real thing?

The theory of computer graphics draws heavily on computational geometry. It studies algorithms for projecting objects onto the viewing surface, removing hidden lines from the projection, ray-tracing, shading surfaces, showing reflections, and rendering translucent surfaces. It has yielded new algorithms for computing geometric forms. It has used chaos theory to create efficient algorithms for generating complex structures resembling natural formations such as trees, coastlines, clouds, and mountains. Graphics theory also uses color theory, which relates colors formed from light on screens to colors formed from pigments on printed surfaces. Sampling theory is used to reconstruct images from noisy data, filter out unwanted effects, and remove spurious patterns caused by displaying sampled data on pixel-oriented screens. Important supporting areas are Fourier analysis, sampling theory, linear algebra, graph theory, automata, physics, analysis, nonlinear systems, chaos.

Models have been essential for practical graphics systems. Extensive studies have yielded efficient algorithms for rendering and displaying pictures including methods for smoothing, shading, hidden line removal, ray tracing, hidden surfaces, translucent surfaces, shadows, lighting, edges, color maps, representation by splines, rendering, texturing, antialiasing, coherence, fractals, animation, and representing pictures as hierarchies of objects. Models for virtual reality and distributed interactive simulation are among the most recent additions.

All the models noted above have been implemented and many are available commercially. For example, graphics algorithms are available in the graphics libraries commonly distributed with graphics workstations. Video editors and sophisticated drawing programs are available for personal computers. Graphics to assist in understanding large scientific data sets are now common as part of high-performance computing, where they are known as “visualization tools”. Color models have been used to produce practical hard copy printers that print with natural hues and agree with the colors on a graphics screen. Graphics standards have been promulgated (e.g., GKS, PHIGS, VDI), along with standard printer languages (e.g., PostScript), specialized graphics packages for individual disciplines (e.g., Mogli for chemistry), and virtual realities on Web pages (e.g., VRML). Image enhancement systems have been used for years; for example, the Jet Propulsion Laboratory regularly released NASA pictures of planets, and 3D visualizers are now available to assist doctors interpret CAT and MRI data.

9 Human-Computer Interaction

This area deals with the efficient coordination of action and transfer of information between humans and machines via various human-like sensors and motors, and with information structures that reflect human conceptualizations. Important contributors to this field are computer graphics and user interfaces. Fundamental questions include: What are effective methods for receiving input or presenting output? How can the risk of misperception and subsequent human error be minimized? How can graphics and other tools be used to understand physical phenomena through information stored in data sets? How can people learn from virtual worlds simulated for them?

Theory in human-computer interaction involves cognitive psychology and risk analysis. Cognitive psychology is important to understanding how humans perceive displays and react; it gives designers the means to evaluate whether humans will misinterpret information presented to them, especially in times of duress. Risk analysis is important because many user interfaces control and monitor complex, safety-critical systems. Important supporting areas are statistics, probability, queueing theory, and coordination theory.

Models and associated measurements are critical to Human-Computer Interaction (HCI). Computer-Aided Design (CAD) systems have come from the application of these models to the domain of design of mechanical parts; a script for running a manufacturing line can be derived automatically from the CAD database. CAD systems incorporate much experience from different approaches to geometric modeling, the efficient representation of physical shapes by computer data structures. Sophisticated image processing and enhancement methods have been developed that allow interpretation of photographs from deep-space probes to human CAT and MRI scans. Principles for designing displays and control panels for ease of use and resistance to human misinterpretation have been deduced from experimental studies.

Design is the central focus in HCI. Usability engineering is a name given to the processes of engineering design for user interfaces. Sophisticated approaches to input, output, interaction, and multimedia have been developed. CAD systems are widely used in manufacturing and computer chip design; small versions of these systems are available for personal computers and desktop workstations. The user interface design arena has evolved a number of popular standards such as icons and menus for display of possible functions and the mouse for use as an input device. New user interfaces built on pen-based computers have come to market and voice-operated computers are not far behind. Flight simulation systems have been used by NASA for years to help train pilots; scaled down versions are available for personal computers. Distributed Interactive Simulation (DIS) systems are regularly used in defense applications to train people how to cope

with battlefield situations; the DIS presents each participant with a real-time image of the world seen from that participant's perspective.

10 Computational Science

This area deals with explorations in science and engineering that cannot proceed without high-performance computation and communications. Computation is seen as a third approach to science, joining the traditional approaches of theory and experiment. It is being used to address very hard problems, sometimes called “grand challenges”. On the computing side, this area deals with general methods of efficiently and accurately solving equations resulting from mathematical models of physical systems; examples include airflow around wings, water flow around obstacles, petroleum flow in earth's crust, plasma flow from stars, weather progression, and galactic collisions. Within computer science, this area was called “numerical and symbolic computation” for many years; since the mid 1980s, it has borne fruit as the many other scientific and engineering disciplines have incorporated computation into their own processes of investigation and design. Fundamental questions include: How can continuous or infinite processes be accurately approximated by finite discrete processes? How can algorithms minimize the effects of errors arising from these approximations? How rapidly can a given class of equations be solved for a given level of accuracy? How can symbolic manipulations on equations, such as integration, differentiation, and reduction to minimal terms, be carried out? How can the answers to these questions be incorporated into efficient, reliable, high-quality mathematical software packages? How can data sets generated by these models be most effectively visualized for human comprehension?

This area makes extensive use of mathematics: number theory deals with finite, binary representations of numbers and error propagation in arithmetic calculations; linear algebra deals with solving systems of linear equations that expressed as matrices; numerical analysis deals with complex solution algorithms and error propagation when they are used; nonlinear dynamics deals with chaotic systems. Supporting mathematics include calculus, real analysis, complex analysis, discrete mathematics, and linear algebra. Other areas of theory contribute here as well, notably parallel algorithms, optimizing compilers, distributed computation, organization of large data sets, automatic discovery in data, computational geometry, graphics (often, in this context, called scientific visualization), statistics. This theory is mingled with the theory in the particular area of science in which a computational investigation is being performed. For example, theories of quantum mechanics are being used to explore a new paradigm of super-fast “quantum computing”.

Computational scientists are avid modelers. They have experimentally-validated models for: physical problems, discrete approximations, backward error propagation and stability, special methods such as Fast Fourier Transform and Poisson Solvers, finite element models, iterative methods and convergence, parallel algorithms, automatic grid generation and refinement, scientific visualization, and symbolic integration and differentiation. As in theory, the models of computing are joined with models from other scientific areas in which a computational investigation is being performed.

Computational scientists have designed many important packages and systems such as Chem, Web, Linpack, Eispack, Ellpack, Macsyma, Mathematica, Maple, and Reduce. They have contributed to models and algorithms in many other disciplines, especially with the "grand challenge" problems such as in physics (e.g., demonstrating existence of certain quarks), aerodynamics and flow dynamics (e.g., numerical simulation of the air flow field around an airplane in flight), chemistry (e.g., designing enzymes and proteins that selectively attack viruses), biology (e.g., joining DNA sequence fragments into the full human genome, microscopy, tomography, crystallography, and protein folding), geology (e.g., predicting earthquakes), astronomy (e.g., locating the missing mass of the universe), meteorology (e.g., long term weather forecasting), earth sciences (e.g., charting the relation between ocean currents and world climate), structural mechanics (e.g., effects of wind and earthquake on stability of buildings, bridges, boats, cars, and planes), electromagnetics (e.g., strengths of fields inside partial insulators, optimal placement of antennas and waveguides, propagation of waves in atmosphere and space), and engineering (e.g., interaction between control surfaces and dynamic stress movements in structures). Massive federal support in the USA for these grand challenge problems has helped not only to solve those problems, but to build large parallel supercomputers and fast gigabit networks.

11 Organizational Informatics

This area deals with information and systems that support the work processes of organizations and coordination among people participating in those processes. Information systems are essential to the success of commerce and business in the growing global marketplace. Because most of the work of organizations occurs in human processes, information systems must be designed with an understanding of human work. Therefore this has been a major area of collaboration between computing people, systems engineering, and people in organization disciplines such as management, marketing, decision sciences, management sciences, organizational systems, and anthropology. The fundamental questions come from the organizational

disciplines, not from computing, but give considerable inspiration to the associated areas of computing.

Many parts of computing contribute theory to organizational informatics, notably languages, operating systems, networks, databases, artificial intelligence, and human-computer communication. Linguistics has provided theories, such as speech acts, which have been used to map work processes. Organizational sciences, such as decision sciences and organizational dynamics, contribute their theory as well. Human factors and cognitive theories from psychology play important roles. Social theories from anthropology have been used to understand work.

Models, abstractions, and measurements are even more dominant than theories in this area. Most of the theories noted above are descriptive; thus models and simulations are used commonly to obtain forecasts.

Management Information Systems (MIS) is a long-standing commercial arena in which computing systems consisting of workstations, databases, networks, and reporting systems are deployed in organizations to assist them in their work. Many decision support systems are available commercially; they range from simulation and mathematical models that forecast market, economic, and competitive conditions to cooperative work systems that assist people in reaching decisions as groups or collaborate together over a network. A new domain of software, workflow management systems, has become a billion-dollar industry.

12 Bioinformatics

This is an emerging area of intimate collaboration between computing and the biological sciences. Investigators are exploring a variety of models and architectures that can revolutionize computing, biology, and medicine. Examples: (1) DNA chemistry has been used to encode and solve combinatorial problems, opening the possibility of chemical computation. (2) New string analyzing algorithms are searching through base-pair sequences in the sprawling network of databases compiled in the Human Genome Project, attempting to construct the overall genome from many fragments. (3) Architects and physicians have produced cochlear implants that restore hearing and prototypes of silicon retinas, opening the possibility of practical, bionic prostheses. (4) Computer analyses are used extensively in genetic engineering to determine the proper chemical structures of enzymes to treat medical conditions. (5) New kinds of organic memory devices are being studied that would be capable of storing data at a thousand times current densities or more.

The Future

The pattern of evolution exhibited in the matrix of twelve subareas and three processes continues. The discipline of computing will experience its most rapid development in the domains of significant overlap with other fields such as computational science, cognitive science, library science, organizational informatics, bioinformatics, manufacturing, and architecture. All the subareas will be infused with new concepts and terminology from the areas of major collaboration as well as from many application domains.

As the discipline has matured, important subgroups from each of the twelve subareas have claimed separate professional identities, formed professional groups, codified their professional practice, and started their own literature and communities. Some of these groups, believing that Information Technology is more inclusive than Computer Science, have started to claim they are part of the IT Profession rather than the CS Discipline. Some of them, such as software engineering, are coming to see themselves as peers of Computer Science within the IT profession. In addition, a number of other IT-related groups have claimed identities within the IT profession. The "IT family", which consists of Computer Science, its children, and its cousins, includes at least two dozen members:

artificial intelligence	knowledge engineering
bioinformatics	management information systems
cognitive science and learning theory	multimedia design
computational science	network engineering
computer science	performance evaluation
database engineering	professional education and training
digital library science	scientific computing
graphics	software architecture
HCI (human computer interaction)	software engineering
information science	system security and privacy
information systems	system administration
instructional design	web service design

Several important conclusions can be drawn from these developments: (1) The IT profession has an enormous scope, including subfields from science, engineering, and business. (2) The players share a common base of science and technology but have distinctive professional practices. (3) The players are willing to identify with the IT field but not with the Computing discipline. (4) Strong leadership from the professional societies will be needed to keep these players united under the common IT identity. (5) The ability of the IT field to resolve broad, systemic problems such as software quality, basic research, and professional lifelong education will require extensive cooperation among the players, cooperation that is endangered if the groups splinter and factionalize. (See Denning 1998).

The subarea of software engineering illustrates the tensions referred to above. As the IT field matures and touches more people's lives, the demand for computing systems to be demonstrably safe and reliable increases. Despite concerted attention to software engineering since 1968, when the "software crisis" was first declared, and despite enormous advances in tools and methods, public dissatisfaction with software systems is at an all-time high. This is manifested in many ways, from widespread complaints about technical support for hardware and software on home and office systems to fears about failures in safety-critical software systems. Public concern has awakened political interest. It is likely that within a decade government licensing of software engineers will be a common practice. In mid 1999, the Council of ACM declined to have ACM participate with any agency in the development of license tests: it felt that software engineering was still immature and that no known certificate can guarantee that the holder was capable of producing safe and reliable software. The IEEE Computer Society disagreed and resolved to participate in the development of licensing tests. Both societies said they want to work toward profession-administered certifications.

The notion of information, which seems central to the discipline, is likely to come under attack. Information is now usually understood as signals or symbols that convey meaning to human observers. Some thinkers have gone further, asserting that information is the common principle underlying physical, biological, human, organizational, and economic systems and, hence, that information science is the parent of all these disciplines. There is, however, a problem with information. Whether information is present is an assessment made by each observer; there are no commonly accepted standards for the assessment (Spinoza et al 1997, Winograd 1996). To many observers, it seems unscientific to claim that computing science is based on a principle that is fundamentally subjective. Some have claimed that the belief that information is a scientific quantity has catalyzed systemic problems in software quality, education, and design (Talbott 1998). (Even Claude Shannon and Warren Weaver, who are credited with formalizing information in their mathematical theory of communication, disclaimed any connection to the way "information" is understood in everyday life; their definition does not apply to the symbols processed by programs.)

The correctness of a program, understood abstractly as a mathematical function, can be assessed based on assertions about the input, intermediate, and output data processes by the program; none of this relies on a formal definition information at all. The correctness of interactions between humans and computing systems, on the other hand, often depends heavily on the assessments people make about the system, and information is an allowable assessment. Thus we would be on much safer ground by claiming

that programs process data and that information is an assessment arising in the interactions between programs and people.

REFERENCES

S. Amarel. 1971. "Computer science: a conceptual framework for curriculum planning." *Communications of ACM 14*, 6 (June).

B. Arden (ed.). 1980. *What can be automated? -- The Computer Science and Engineering Research Study*. Cambridge, MA: The MIT Press.

P. Denning, D. E. Comer, D Gries, M. C. Mulder, A. Tucker, A. J. Turner, P. R. Young. 1989. "Computing as a discipline." *Communications of ACM 32*, 1 (January), 9-23.

P. Denning and R. Metcalfe. 1997. *Beyond Calculation: The Next 50 Years of Computing* (P. Denning and R. Metcalfe, eds.), Copernicus Books, an imprint of Springer-Verlag.

P. Denning. 1998. "Computing the Profession." *Educom Review 33* (Nov-Dec), 26-39, 46-59.

P. Denning. 1999. *Talking Back to the Machine: Computers and Human Aspiration*. Copernicus Books, an imprint of Springer-Verlag.

P. Drucker. 1989. *The New Realities*. Harper.

R. W. Hamming. 1969. "One man's view of computer science." ACM Turing Lecture. *Journal of the ACM 16*, 1 (January), 1-5.

J. Hartmanis et al. 1992. *Computing The Future*. National Academy of Science Press.

National Academy of Sciences. 1968. *The mathematical sciences: A Report*. Publication 1681. Washington, DC.

C. Spinoza, F. Flores, and H. Dreyfus. 1997. *Disclosing New Worlds*. MIT Press.

S. Talbott. 1998. "There is no such thing as information." *Netfuture 81*. Available from <http://www.oreilly.com/~stevet/netfuture>.

A. Tucker, Jr., and P. Wegner. 1996. "Computer Science and Engineering: The discipline and its impact." In *Handbook of Computer Science and Engineering*, CRC Press, Chapter 1.

P. Wegner. 1970. "Three computer cultures -- computer technology, computer mathematics, and computer science." In *Advances in Computers 10* (W. Freiberger, ed.). New York: Academic Press.

T. Winograd. 1996. *Bringing Design to Software.* Addison-Wesley.

T. Winograd. 1997. "The design of interaction." In *Beyond Calculation: The Next 50 Years of Computing* (P. Denning and R. Metcalfe, eds.), Copernicus Books, an imprint of Springer-Verlag. 149-162.