UMEÅ UNIVERSITY
Institute of Information Processing – ADB

Postal address:	Tel (direct dialing):	Telefax:	Email (Internet):
S-901 87 UMEÅ (Sweden)	+46 90 166030	+46 90 166126(166688)	kivanov@cs.umu.se

Professor KRISTO IVANOV
Chairman, Administrative Data Processing

Draft, 31 August, 1989

SOFTWARE SYSTEMS

The basic issue

Software and hardware are two "primitive" categories of computer science and as such they are often taken for granted in discussions of research matters. They share this position with other primitives like syntax, semantics and pragmatics.

It is probable that in the context of our research *computing* should be considered as a more primitive category, and it is remarkable how little reflection is found on this matter, with few notable exceptions (Davis & Hersh, 1986, pp. 139-164). Many people would not hesitate today to equate the art of programming with the vaguely defined young discipline of "datalogy" (Naur, 1968). Software, in any case, is distinguished for being considered by many to lie at the center och computer science. This is evidenced by the often repeated statement about the computer being "dumb" and powerless without a program which endows it with intelligence and with life. As a matter of fact things are not so clear since it could also provocatively be claimed that a computer program without an intelligently designed and manufactured computer (hardware) is dumb and powerless (Ivanov, 1983). At the bottom of this issue probably stands a reductive vision of reality by which it is reduced to logic and physics (logical empirism), and it is translated into the corresponding terms of software and hardware. If this reduction is illegitimate it will show up in the form of vagueness of the delimitation of the software concept, the rise of such culturally absurd concepts as "courseware" (Schwartz, 1986c), and a general intractability or impossibility for rigorous evaluation of the applications of research results.

The basic issue in the given perspective is, in any case, whether an improvement of the performance of a social group or and organization in a particular case is better attained through better programming, through the development of a new programming language or through the reallocation of individual responsibilities and change of work routines. In order to answer this it is assumed that some knowledge is necessary about the interface between organization theory and programming theory, if they are to be considered as different theories at all. In some approaches they are namely both considered to be inquiring systems, dealing with basic concepts of systems and organization. In this respect it will also be recalled that mathematics and logic or symbol manipulation, including visual symbols, have classically been considered to be a basic philosophical language for discussing and developing inquiring systems.

In what follows we will shortly mention some fundamental problems which have been outlined in the area of computer software. We will then describe several research ideas that have been proposed and sometimes also implemented in order to prevent or solve software problems. Since these ideas are difficult to relate to each other and to consider in a whole perspective they will be catalogued as constituting an eclectic approach. With the purpose of enhancing wholeness and total understanding some historical insights will then be presented, followed by an outline of proposed research directions including practical experimental or applied approaches. The question of the relations among better programming, programming languages, and reallocation of individual responsibilities or change of work routines will be hinted at in terms of points of contact with research on mathematics, logic, and social organizational systems.

The problems

During the decade of the 80's several observations have been made about what may be called the software crisis including the corresponding educational crisis. (Bubenko, 1988; 1989a; Parnas & Clements, 1986; Roosen-Runge, 1989; Schwartz, 1986d; Wasserman & Gutz, 1982; Weizenbaum, 1976; Winograd, 1979).

Let us review some views that have been formulated in the pertinent scientific community:

Computer programming today is in a state of crisis or, more optimistically, in a state of creative ferment. (Winograd, 1979).

Since lack of firm understanding of how to proceed with software has never fully restrained the drive to increased functionality, huge, sprawling, internally disorganized and externally bewildering systems of overwhelming complexity have already been built and have come to plague the computer industry. The situation is ameliorated in the short term, but at length exacerbated, by the economic 'softness' of software: costs are incurred incrementally rather than in a concentrated way; no single detail is difficult in itself, but the accumulation of hundreds or thousands of discordant details makes the overall situation impossible. (Schwartz, 1986d, pp. 107.)

Hierachical decomposition of business activities or business functions (process or "data transformation" oriented methodologies), as well as of data or information flows (data oriented methodologies), have been advocated and used since the sixties for the development of computer systems. Hierarchical decomposition is a well accepted as a practical tool and it has an accepted standing in research (Olle, H.J., & Verrijn-Stuart, 1986; Olle, Sol, & Tully, 1983; Olle, Sol, & Verrijn-Stuart, 1982). It is therefore astonishing that, during more than 20 years, relatively little formal theoretical work has been presented concerning basic concepts and definitions of this technique. Furthermore, the relationship between the process oriented techniques and the data oriented techniques, such as conceptual modeling (Bubenko, 1983; Humphreys & Berkeley, 1988; Sowa, 1984), is still far from clear. (Bubenko, 1988).

The programming community has long recognised that we could be using computers in the production of software products more than we now do. Recently computer aided software engineering "CASE" has become another in a long series of buzzwords in our field and large, money making, conferences are held where vendors tout their tools. Strolling the lanes of these expeditions is somewhat disappointing. The vast majority of the tools being offered are not based on a coherent software design process and have no basis in the semantics of programming.... We examine a "rational" design process, identifying models that provide insight into what we are doing in each stage, and propose ways that CASE tools could help. While we have no tools for sale, we believe that these mathematical models will lead to high quality tools and an improvement in the standards of software engineering. These mathematical models are not those usually discussed by Computer Scientists, but are closer to those used in other fields of engineering. (Parnas, 1989b)

For over 30 years, the central dogma of computer science has been that computer users require programmers who can compose programs in a programming language. But despite three decades of research and a Babel of programming languages, we still confront substantial and quite unnecessary obstacles in producing reliable non-trivial programs at low cost, for our programming languages fail to take into account essential facts about the cognitive and social aspects of computing. (Roosen-Runge, 1989.)

Nearly a quarter of a century later, I have, reluctantly, concluded that our opponents were right. As I look at computing science departments around the country, I am appaled at what my younger colleagues, those with education in computing science, don't know. Those who work in theoretical computing science seem to lack an appreciation for the simplicity and elegance of mature mathematics. They build complex models more reminiscent of the programs written by bad programmers than the elegant models that I saw in my mathematics courses. Computing scientists often invent new mathematics where old mathematics would suffice. They repeatedly patch their models rather than rethink them, when a new problem arises. Further, many of those who work in the more practical areas of computing science seem to lack an appreciation for the routine systematic analysis that is essential to professional engineering. They gravitate toward the flashy or exciting areas in which programs are written, tested, elaborated, demonstrated, and then discarded. They eschew the systematic planning, documentation, and validation that is essential in engineering. In violation of the most fundamental precepts of engineering design, some "practical" computing scientists advocate that implementors begin programming before the problem is understood. Discussion of documentation and practical issues of testing are considered inappropriate in most computing science departments. In traditional engineering there is a cooperation between theory and practice....Most of what is taught in computing science "theory" courses is not relevant in the practical areas. Much theory concentrates on machines with infinite capacity, but such machines are not available to practitioners. (Parnas, 1989a.)

Most of what is reported above is confirmed by our own limited experience. There are also distressing reports about shortcomings and failures of information systems and expert systems (Buchanan, 1986; Lyytinen, 1986, pp.1-14; Sørgaard, 1986; Östberg, Whitaker, & Amick III, 1988) as well as about the overwhelming complexity of the problem (Ivanov, 1972; Ivanov, 1988; Lyytinen, 1988a; Lyytinen, 1988b). To the extent that such findings can be questioned, that will itself be an illustration of the pro-blems which are mentioned, where a chaotic multiplicity of approaches and specializations, and an uncontrollable empirical trial-and-error heuristics, eschew any serious experimental overview and evaluation. As already pointed out, the concept of experimentalism or controlled empiricism itself may not be applicable to an epistemologically problematic area where replications are not possible because of technological obsolescence, there is an interplay of strong pressure groups, etc. (Churchman, 1979). A theoretical and historical understanding of earlier empirical efforts in the design and evaluation of software could facilitate previsions of the results of new efforts and guide the design and application of new programming languages

Eclectic approaches

Whenever the unitary view of a research field is lost there will be a temptation to establish a taxonomy or classification scheme as a preliminary orientation. The often unstated assumption seems to be that a certain "type" of problem is to be solved through the use of a certain type of programming language that is chosen, as it were, like a tool out of a tool box. One unstated epistemological background such tool-thinking is probably the well known mean-ends scheme of thought (Churchman, 1961; Ivanov, 1988). Current taxonomies or internal partitions of tool boxes use sets of more or less equivalent and mutually inconsistent, non exhaustive denominations for families of programming languages such as 1) Procedural, imperative or transition oriented, 2) Object oriented and simulation oriented, 3) Logic, functional, value-oriented, declarative or applicative, 4) Production-rule oriented, versus Constraint oriented, 5) Concurrent, versus Sequential von Neumann types, 6) Data-base query types, or 7) Mixed, combinations of the above.

On different occasions one may find attempts to formulate overviews (Kay, 1984; Nilsson, 1987, pp.5-8; Schwartz, 1986d; Yeh, 1977), or in depth discussions of procedural languages (Wirth, 1976), object oriented languages (Robson, 1981), object oriented versus value oriented languages (Harland, 1984), logic languages (Kowalski, 1979), parallel computation and concurrent programming (Ben-Ari, 1982), or of applicative or functional programming (Henson, 1987; Richards, 1985, Backus, 1978 #38). Visions about mixed or combinated languages have also been advanced (Wasserman & Gutz, 1982).

It seems that the latest trends in terms of taxonomies tend to confuse such already confused taxonomies in the sense that there is a development towards the 7th category above, of mixed combinatory approaches. This is particularly encouraged by the increased emphasis of talk about artificial intelligence, expert systems, mixed expert and database systems, hypermedia, etc. The trend is partly covered also by the label of CASE, that was challenged for being mainly a new marketing "buzzword" launched by oversellers in order to cover up the felt need for order out of chaos. In any case it is obvious that such unfolding challenges the understanding of what software and software engineering is all about when an unholy melting pot of AI and databases is assumed to revolutionize the art of programming (Berztiss, 1988, may be regarded as a symptom of this unrest). The announcement, for instance, of an advanced course in AI in the autumn 1988 on the campus of a Swedish university, formulates its intention to "enlarge, deepen and structure the fundamental paradigms and approaches to knowledge representation and reasoning in AI", i.e. to develop new concepts of programming. "This is to be achieved first by providing a structured historical perspective of the development of three major approaches, which are more or less in conflict with each other, though sharing some fundamental concepts: 1) Cognitive-oriented AI, 2) Logic-oriented AI, and 3) Procedurally-oriented AI". In this way an additional problematic "cognitive" type is being added to the above mentioned taxonomic categories, with the belief of having been able to identify some new and fruitful fundamental shared concept.

The hope that some order out of chaos can be attempted through an application or the classical procedure of axiomatization (Hoare, 1969), prompts an unending stream of tentative unrelated axiomatizations for new "languages" that have unclear objectives. Ad-hoc "arbitrary" definitions will then be made for e.g. objects, concepts, entities, relationships, relations, attributes, associations, events, triggers, conditions, states, structures, etc. (Bubenko, 1983, pp.9-10).

Such definitions should, however, be consistent with other researchers' earlier conceptualizations. One of them (Berild, 1981) proposes e.g. that *statement about a phenomenon* as be composed of *objects* such as *event* (e.g. "employment"), *effect* of event [=action?] (e.g."employee"), *participant/generator* of an event (e.g. "employer"), *process* (e.g. "hiring interview"), and *measurement/opinion* (e.g."skill-certificate").

No notice was taken above of a closely related earlier set of primitives in the spirit of logical empirism which was launched in the context of defining an elementary information message (Langefors, 1973). In any case, later on a new set of basic concepts with the use of the same words happened to be used in the context of computerized support in mechanical robotics where formalisms are developed for the representation of so called *action-structures* Bäckström, 1988; Sandewall, 1986 #636]. The basic ideas of the approach was 1) An *action-structure* as a set of actions each of which has a starting and ending point, the set of all such points being partially ordered in time, 2) Partial *state descriptions* , 3) *Pre-condition* and *post-condition* which characterize what must hold at the beginning and end of the action respectively (and are sufficient to express when two actions are *allowed to,* but are not sufficient to express when they *have to,* be executed in parallel without interfering), and 4) *Prevail-condition* which characterizes what must hold for the whole duration of the action. In this way it appears that the proposed ad-hoc formalism implicitly attempts to construct a formal system theory which in the computer replaces the kinematics (space plus time) and dynamics (plus force) of classical Newtonian mechanics. This very same "epistemological" operation is, by the way, implicitly performed by another author (Janlert, 1988, ch.7) in analyzing the "desktop ontology" of the Macintosh man-machine interaction. It is therefore symptomatic that the approach states that the use of prevail-conditions may be ambiguous, since it can express both what an action *requires* to hold and what it *causes* to hold during its execution. An example of the former is then given as that the action of "frying an egg" *requires* the stove to be hot during the whole execution of the action, while an example of the latter use of prevail-condition is not as well elaborated but it could be the case of a "car-driving action" which could have as a prevail-condition that the car is kept on the road, a condition that surely is under the control of the car-driving action (Bäckström, 1988, part II, p.4). This could be compare with applying the LOGO language to describe the process involved in learning the physical skill of juggling (Papert, 1980 p.105).

The case described above is interesting from the research methodological point of view since it evidences the introduction of categories of action, structure, state, conditions, cause, requirements, etc. which have unclear relations to the formalism of classical Newtonian mechanics or, for that matter, to classical formalisms of programming. They disclose also famous difficulties of Newtonian determinism as related to teleology, as it may be evidenced by the almost direct comparison with an example of frying eggs on the stove (Churchman, 1971, chapter 3.) It is also symptomatic that the proposed kind of "systems programming" language of action structures works with syntactical and semantical considerations but not with pragmatic ones, in spite of being avowedly concerned with *actions* (but not *actors*). Here the modern formalistic definitions of syntax and semantics may themselves be questioned (Stenlund, 1988, pp.73-74).

The same can be remarked about more recent "object management systems" (Björnestedt & Britts, 1988), where *implementations* and *operations* are added to other more traditional primitives. In last analysis these kinds of search for fundamental systematization of such things as "object systems", "partial world models" or frames, and "3-valued logic" (Bäckström, 1988, II, p.10) probably will lead to a late rediscovery of "inquiring systems", the issue of separability of subsystems and logic of uncertainty or languages of doubt (Churchman, 1971, pp. 54ff., 103-155). That would illuminate the confusion of those who, examining e.g. the issues of "hierarchical decomposition" of business activities and data flows, feel that it "uncovers several obscure and unclear issues, and raises many problems and questions" (Bubenko, 1988).

In a similar vein, the development of natural languages for user dialogue and for database inquiries requires "languages which can handle dialogue" (Jönsson, 1987, p.12), and assumes that components of natural language interface will include a lexicon, morphological analysis, syntactical analysis, semantical analysis or meaning without context, and pragmatic analysis or meaning with context (ibid., p.5). The kinds of questions which are expected to be answered by the participants of a dialogue turn out, once more, to be related to the understanding of what is to be considered an object, its identification, its relations in valid structures with attributes, and the wider context. Such concepts recall the framework which has been advanced in an apparently quite different occasion, when dealing with the structuring of user dialogues (in a loose and wide sense) about statistical information, and requiring such categories as *activities, performance, environment,* and *entity* (Dunn, 1974; Ivanov, 1976; 1979; 1984).

All this points to basic insufficiencies in the understanding of interactive dialogue and programming as well as of the concept of dialogue and inquiring system beyond purely formal conceptions of the nature of logic (Levin & Moore, 1988). In a special proposal concerning research on logical aspects of computer and information systems we will mention some struggles about this question which have being going on in more and less formal terms (Fischer, Morch, & McCall, 1989; Fisher, 1987; Forsgren, 1988; Freedle, 1975; Levén & Nordström, 1989; Pask, 1976).

There are, finally, some eclectic approaches to software that recommend themselves for the exemplary unpretentiousness of their theoretical claims and their down to earth engineering anchorage with sound diffidence of empty formalisms (Hester, Parnas, & Utter, 1981; Parnas, 1972; Parnas, 1976; Parnas, 1985; Parnas & Clements, 1986; Parnas, Clements, & Weiss, 1984; Parnas, van Schouwen, & Kwan, 1988; Utter, 1982; Utter, 1984). They seem to work in the same systematic engineering spirit as the Scandinavian "theoretical analysis of information systems" (Langefors, 1973). They seem to be best represented by a recent statement of an ideal educational program for software engineers (Parnas, 1989a) that should be contrasted with earlier grand front attacks of the educational issue (Computer science curriculum, 1964; Denning, Comer, Gries, Mulder, Tucker, Turner, et al., 1989; Forsythe, 1967; Forsythe, 1970; Information systems-curriculum recommendations of the 80's, 1982). Their eclecticism, however, sound as it may be, still does not allow to see the bridge over to the educational ideas that have been formulated in the social systems field (Gharajedaghi, 1986).

<u>History: the roots</u>

Whenever concern for foundations is not allowed in the research process there will be the risk of irrefutable empirical doing, and an ad-hoc common sense philosophizing which unconsciously attempts to simulate the historical dialogue between the best minds of Western philosophy and science in the past 50, 100, or 500 years. Lack of unity or of explicit relationships between different approaches prevents the much advertised accumulation of knowledge which is expected in modern research. It may be one practical consequence of the unrestricted ad-hoc pluralism which followed the so called foundational crisis of modern formal science which is embodied in the computer and in computer networking. The crisis of foundations may be interpreted at various levels of sophistication. What can be said at this point is that inquiry about what programming and models of computation are or should be in relation to the "nature of machines", or inquiry about the alleged difference in the meaning of functional specifications for programming versus for artificial intelligence, etc., opens up an great quantity of important and questionable issues which are today buried under the apparent gray simplicity of everyday's practical scientific, or rather technical, work (Doyle, 1982, pp.11-16). This throws us into questions of metamathematics (Kleene, 1950; Webb, 1980) which probably are too abstruse for most researchers who are working in computer and information science. Such questions, which may already be "too difficult" because of being "corrupted" by an unhappy development of modern mathematics could also be more successfully attacked at the level of a cultural debate about the meaning of the quest for the foundations of mathematics (Brouwer, 1905; Stenlund, 1988; Weil, 1970-1974; Weyl, 1949; Zellini, 1988) or at the "biographical" level (Hodges, 1983; Reid, 1970; van Stigt, 1979).

It has been remarked (Rota, 1986d) that an unbridled and passionate interest in foundations has often been singled out as a characteristic trait of both philosophy and science in this century. Nowhere has this trend been more rampant than in mathematics. Yet, foundational studies, in spite of an auspicious beginning at the turn of the century, far from achieving their purported goal, found themselves attracted into a whirl of mathematical activity. As mathematical logic becomes ever more central within mathematics, its contributions to the philosophical understanding of foundations wane to the point of irrelevance. Worse yet, the feverish technical advances in logic in the last ten years have dashed all hope of founding mathematics upon the notion of *set,* which had become the primary mathematical concept since Cantor (ibid., p. 167).

If this is so in mathematics, it is to be expected that the foundational problems will be much worse in a research area like programming, with so unclear connections to its closest disciplines of logic and mathematics. Mathematicians (Rota, 1986e) have occasionally expressed the opinion that important problems concerned with programming languages and related questions of software do not easily fit into existing disciplines and that if anywhere they belong somewhere between linguistics and logic. On the other hand problems concerning algorithmic complexity, which unquestionably belong to mathematics, could have been formulated before computers came into existence. In views of the future of computer science with emphasis on robotics it has been envisaged (Schwartz, 1986d) that one will have to go beyond the numeric combinatorial, and symbolic manipulation that have been the main concern till now: computer science must confront the geometric, dynamic, and physical realities of three-dimensional space. This confrontation will greatly deepen the foundational connections of programming to classical applied mathematics and physics. The earlier mentioned difficulties in attempts to represent action structures (Bäckström, 1988), based on earlier conceptual work (Sandewall & Rönnquist, 1986) may be a symptom of insufficiently deep connections to physics. Obviously the difficulties will be even greater whenever action structures would try to embrace social actions as found in administrative and expert systems applications.

The crisis of foundations is not limited to well established sciences such as physics and mathematics, but it reappears under different guise in the life sciences and the social sciences. We are reminded (Rota, 1986d) of Edmund Husserl's thesis that the culprit for the relative lack of success of these sciences is to be found in the hasty adoption of logic and methods of physical sciences without going through an independent Galilean process of concept formation. The very need to match the rigor of physical science was, and is, erroneously replaced by an uncritical imitation of physical methods and techniques. This is a danger that is exemplified by certain strivings for premature introduction of formal languages in computer science (Bubenko, 1988, cf. the quotation above), and has been insightfully noted in recent work in mathematical biology (Rosen, 1985a; Rosen, 1985b). In the realm of mathematical economics, close to administrative and geographical information systems, heavy criticism has been directed against "symbolic pseudomathematical methods" (Schwartz, 1962) as well as against their disregard of error-theoretical aspects (Morgenstern, 1963).

It is then clear that we will have difficulties in unearthing the foundational roots of programming and software systems, but the attempt must be made.

To begin with, it is intuitively attractive to consider that software development or programming is scientifically based on a Cartesian (Barrett, 1987), and, further, a Newtonian world view, a Newtonian mathematics and logic, particularly Boolean and symbolic mathematical logic. These kinds of logic should be considered in their relations to mathematics. They were early integrated with physics as represented by the mechanics of the industrial revolution as evidenced in the often mentioned programming of Jacquard's loom. Later they would be integrated in physics as represented by electrical engineering, particularly digital electronics. In these contexts various historical overviews could be helpful (Augarten, 1984; Encyclopedia of computer science and engineering, 1983; Evans, 1981; Pylyshyn, 1970; Randell, 1973), as well as "biographies" of people and products (Hodges, 1983; Jobs, 1989; Kidder, 1982; Lammers, 1988; Rheingold, 1985) including the latest stories about the life and deeds of successful programmers. A particular necessity is to consider the necessity of relating mathematics and logic not only to physics and engineering but also to their economic and industrial aspects, as these aspects appear, for instance, in the Marxistic tradition (Annerstedt, Forssberg, Henriksson, & Nilsson, 1970; Mendner, 1976; Quiniou, 1971).

One important root of software thinking lies in the hardware (Davis & Hersh, 1986, pp. 165-178, advances a couple of opinions in this respect). It is an idea that appears most explicitly in the context of microprogramming. Historically it may be grounded in certain pioneering work of digital electronics where Boolean algebra of logic is used as a symbolic method of investigating logical relationships. Arithmetic and Boolean logic are properly acknowledged to form the computer's basic reflexes (Schwartz, 1986b), even if two basic ideas, namely Charles Babbage's stored program concept and its integrated microelectronic implementation (built up by the diffusion of electric catalysts into chips of materials like silicon or germanium) continue in technical terms to drive the computer field as a whole (Schwartz, 1986d).

Boolean algebra of logic was used in the form of calculus of propositions seen as analogs or isomorphs of digital switching circuits (Shannon, 1938). The purpose was "network synthesis", e.g. for finding the simplest circuit containing only series and parallel connections and, in some cases, for finding the simplest circuit containing any type of connection. Up to the 1940s, switching circuits of telephone exchanges and calculating machines were designed by loose methodological principles extracted by trial and error from praxis, with no underlying mathematical theory. The relevance of mathematical logic in this application area was shown only later in terms of the theory of automata or finite state machines, Boolean algebra, Boolean functional equations, etc. This, by the way, has prompted some mathematical systems scientists (Klir & Lowen, 1989) to assume that "soft systems methodology" (Checkland, 1981; 1988) for a particular type of problems develops from praxis, when no mathematical theory is readily available for dealing with the problems at hand: if and then mathematics becomes sufficiently developed for dealing with those problems, the soft methodology is supposed to evolve into the corresponding hard methodology which, as a rule, is more powerful and less experience-dependent than the former.

A discussion of this hypothesis must be left to the research on mathematical aspects of computer and information science. At this point it may suffice to remark that Boole developed his algebra in 1847 as an attempt to lay the foundation of a mathematical theory of logic or to base mathematics on irreductible logical principles, an attempt that was further developed by B.Russell's and A.N. Whitehead's "Principia Mathematica". A curious inversion, however, happened when it was discovered that the structures of Boole's algebra were associated to structures that are peculiar to the theory of numbers /theory of groups, the algebra of residuals modulus 2. It was at the beginnings of this century that the insight was gained that both the "intuitive" algebra of sets of Cantor and the propositional logic of Frege and Russell were isomorphs with Boolean algebra (Katz, Doria, & Lima, 1975, pp. 50-51). It would be interesting to explore the implications of having founded, through the computer, so much of human inquiry on "programmable" foundations that are not foundations and the limits that this puts on what can be obtained through higher level or tool-kit programming. If there is a fundamental "genetic defect" in our basic concept of reason (Barrett, 1987, p. 46; Davis & Hersh, 1986, p. 292; Pirsig, 1974, chaps. 10, 14, and 21) then there may be limits to what can be obtained or avoided with it.

In the present historical context of software aspects we will only complete the above image of the raise of computer programming by observing that the subsequent early attempts to see programming as a unitary scientific field of endeavour displayed an explicit attachment to logic and mathematics. Discussions were hold in terms of heading such as "a common language" for computers (Bauer & Samelson, 1960; Gorn, 1960), or the syntax and semantics of a proposed "international algebraic language"(Backus, 1960), or "a universal language" (Poyen & Vauquois, 1960), or a "general problem solving" program (Newell, Shaw, & Simon, 1960). Boolean logic debouched into more sophisticated forms of mathematics, logic and mathematical logic adapted to descriptions and manipulation of the computer at the machine level, and further to layered concepts or meta-languages (Davis & Hersh, 1986, pp. 132ff.). It corresponds to a sequence starting from microcode, to operating system, assembly language, higher level languages, special purpose software packages, and finally even further up to the CASE idea of computer aided software engineering that tries to encompass even systems analysis of business activities (Bubenko, 1988; Grindley, 1966, is an interesting historical counterpart of Bubenko; Humphreys & Berkeley, 1988, is to our knowledge, together with Wood & Doyle, the most ambitious attempt up to this date; Wood & Doyle, 1989). The business is then seen as a conglomerate of Newtonian objects and properties connected through logical relations and are to be programmed to perform useful activities.

The historical development up to today's long run visions is extremely complex, especially if it is to be related to the scientific, economic and sociopolitical context but there are some valuable more or less popular technical overviews that may constitute a start in such an investigation (Bowers, 1977; Davis & Hersh, 1986, pp. 132ff, 165ff, 179ff.; Elbourn & Ware, 1962; Grosch, 1977; Kac, Rota, & Schwartz, 1986, pp. 49ff, 63ff, 105ff, 207ff; Orchard-Hays, 1961; Strachey, 1966). Even purely technical dissertations include sometimes insightful discussion of development and trends (Lunell, 1983).

By now it is clear that the number and types of programming languages have grown in keeping pace with the technology and with the number of problem areas or areas of application. The hope and vision of a common universal language has given place to an extremely complex pluralism that contrasts the earlier spirit of search for foundations of mathematics and unification of sciences. This pluralism or "polytheism" is paradoxically completed by the "monotheistic" concept of CASE that is supposed in a certain sense to unify the pluralism of languages and tools. The pluralism has in later years also prompted a reaction in the conceptualization of what programming is supposed to be (Nygaard, 1988; Winograd, 1979; Winograd & Flores, 1986). The concept itself of programming gave place to that of software engineering, and further, to that of integrated development environments or problem solving environment in the spirit of CASE, and finally to that of interactive cooperative systems development (Sørgaard, 1988).

An interesting tendency has consequently been the blurring of the distinction between programming, programming languages and systems or user application development. This has been experienced both from the point of view of practitioners of operations research (Stevens, 1982), and of artificial intelligence (Winograd, 1979). Greater attention has been paid to linguistic and phenomenological issues (Cohen & Perrault, 1979; Goldkuhl & Lyytinen, 1982; Winograd & Flores, 1986). This approach to linguistic aspects contrasts with logical empirical linguistic approaches that traditionally dominated the computer field (Bar-Hillel, 1973; Bar-Hillel & Carnap, 1953; Gaines & Shaw, 1984) and influenced the conception of what is to be understood as information and information system (Langefors, 1973).

The above mentioned blurring of distinctions has most forcefully been formulated from within the tradition of programming where the emphasis has been on "general system description languages". It was done with the idea which was launched some years ago in the context of initiating a large cooperative project between Norway, USA, and Sweden, the SYDPOL project for systems development through professional languages. It stated the intention of shifting the research attention: 1) From programming languages *to systems description languages and profession oriented languages,* 2) From staged development *to integrated development and use,* 3) From programming environments *to systems development environments,* 4) From data *to communication,* and 5) From expert systems *to collaborative accumulation of knowledge.*

Some of the results of this effort have recently led to the further goal of using the insights that were obtained from early simulation languages in order to contribute to the creation of new programming concepts, incrementality, and new kinds of user interface (Holbaek-Hanssen, Håndlykken, & Nygaard, 1985; Nygaard, 1988). Such approaches (Nygaard, 1988, in particular) are intended to rely on a deeper understanding concepts such as *phenomenon, object, action-transition, process, structure, state,* as well as "qualities" of information processes and information systems such as *substance* (e.g. the class declaration: records, files), *properties* (with values or quantities, the type declaration), *transformation* (transition, action, the procedure declaration). These considerations are complemented by a list of *functional roles* in system development and use: rulers, managers, operators, customers, bystanders, designers, programmers, and teachers (Churchman, 1979, offers a good basis for an analysis and refinement of these concepts). Instead of developing application or job oriented (computer) languages the striving is to obtain designed extensions of a profession's (natural) language, or profession oriented computer languages which integrate information technology in the particular profession.

Such "transition oriented programming", by the side of object oriented and constraint oriented (functional and logic) programming, have the merit of evidencing some of the complexities which *turn programming and software systems into inquiring systems.*

Research directions

We mentioned above that "transition oriented programming", by the side of object oriented and constraint oriented (functional and logic) programming, have the merit of evidencing some of the complexities which *turn programming into inquiring systems development.* The issue of design of good programming languages will melt down with the issue of good use of these languages, with systems theory, and with the organization of the software engineering process.

At a level of common sense this could be interpreted as a question of designing a good "environment" for software development, a good axiomatized "system" based on adequate primitives such as objects, entities, attributes, relations, states, operations, actions, processes, transitions, transformations, procedures, structures, functions, or whatever superficially precise concept that could be suggested by the text above. Eventual success could be seen as depending upon the type of problem as related to the genesis and appropriateness of various taxonomies of programming languages, programming styles, cognitive styles and associated knowledge representations.

Nevertheless we prefer to dwell upon the "metamorphosis" of programming into inquiring systems that can be seen relatively easily by following some of the work done in transition oriented programming. It will be one main hypothesis in our research that a greater part, if not all, of the software complex can be analyzed and improved by regarding it as a matter of inquiring systems.

Inquiring systems is by now a quite established research field where several of the above mentioned concepts and qualities have been carefully anchored in traditional debates about scientific method. Such concepts are for instance morphological, functional and teleological classes, structure, states and transitions (Ackoff & Emery, 1972, chapters 1-4 and 15, particularly their conception of action versus response; Churchman, 1971, ch.3 and 10), and professional expert decision versus cooperative work (Churchman, 1971, ch.9). These approaches may be favourably compared with some ones in other traditions (Cohen & Perrault, 1979; Goldkuhl & Lyytinen, 1982; Macmurray, Ewing, & Franks, 1938; Orci, 1983, chapters 2,3,5).

It should be obvious that there are definite implications of how basic terms or primitives of a branch of science are conceptualized, and it may make a real practical difference how, for instance, such terms as state, stimulus-trigger, action and therefore reaction and interaction, etc. are defined when they happen to be defined at all. It is natural to expect, for instance, that a theoretical differentiation between action, reaction, and response (Ackoff & Emery, 1972) will make a real difference for the researcher's understanding of what is to be meant by inter-action as opposed to re-action. This will have consequences in the context of planning, performing or evaluating experimental studies of man-computer interaction, dialogue or "logic of questions and answers" (Belnap & Steele, 1976; Freedle, 1975; Pask, 1976; Pask & Gregory, 1987).

There is one implication of the blurring of the distinctions between software and software environment that may historically explain the raise of not only CASE but also that particular class of high level programming languages associated with the prefix HYPER. At fist glance hyperprograms eschew classification in terms of the more detailed taxonomies mentioned above, if they are not classed in the rather meaningsless generic class of "high level" languages. Like some of the newly proposed methods for systems development they do not at first sight relate to mathematics (functions) or to logic (inference networks). The so-called hypertext idea relates rather to the establishment of connections, linkages, chains and networks of associations (a manual "connection machine") among different kinds of data types and data structures. One of objectives of the research program will be to identify the scientific status of such approach and expectations. One hypothesis is that, instead of representing a unprejudiced search for comparisons and contrasts, or "isomorphisms" (van Gigch, 1988), it is an variant of the classic oldfashioned principle of associative learning" (Guthrie, 1942), and of simple empiricism in terms of some of its basic tenets, e.g. proximity, frequency, similarity, etc. (Rychlak, 1977, pp. 18, 107-8, 290, 332). That would be in the same spirit as "Lockean inquiring systems" (Churchman, 1971, ch.5), Mill's cannons of induction (Churchman, 1948), and "precedence analysis" (Grindley, 1966, as an early USA parallel to Langefors; Langefors, 1973), adapted to the new available technology.

Such "chains of associations" or chains of information could also be related to the metaphor of "information flow" and "communication channel" in order to ascertain to what extent the recent ideas of "mapping knowledge flows in enterprises" (Whitaker & Östberg, 1988; Östberg, et al., 1988) can be based on a teleological normative design of organizational information systems instead of relying on an objectivation of information with a consequent enhancement of descriptive bureaucracy (Churchman, 1971, refers to "Singerian inquiring systems" instead of "Lockean inquiring systems").

Associative "hyperchains", on the other hand might be classified in the class of graphs that in turn could be fruitfully analyzed in terms of "Leibnizian inquiring systems" or "fact nets" that can be expanded to accomodate inductive experimental thinking of neural-net type (Churchman, 1971, chaps. 2 and 5). Disregarding for the moment important speculative aspects about the limitations of our logic and conceptual models being patterned exclusively upon the structure of physical objects (Rota, 1986d, p. 169), and about limitations of logical-causal interactions in material terms (Rota, 1986c, p. 177), graphs enjoy a reputable standing as a subject of computer science, and especially in artificial intelligence concerning expert systems (Schwartz, 1986a, p. 185; Schwartz, 1986d, p.114). Their epistemological status, however, is often oversimplified when many problems are forced into the formulation of finding a path between two known points within a graph. As in deduction of theorems in computerized predicate logic, "graph search methods that work beautifully for monkey-and-bananas tests involving a few dozen or hundred nodes fail completely when applied to the immense graphs needed to represent serious combinatorial or symbolic problems" (Schwartz, 1986d, p. 115). There is obviously something that is not proper in regarding the transformation of graph into path as a general formal mechanism which permits to derive through a graph search something structured, to wit a path, from something unstructured, namely its underlying graph. "Any collection of transformations acting on a common family of states defines a graph, and discovery of a sequence of transformations which carries from a given to a desired state can therefore be viewed as a problem in graph search" (ibid. p. 114) is, therefore, a sentence that must be qualified in terms of structure-function and morphological-functional-teleological classes (Churchman, 1971, chap. 3; Singer, 1959). Only in this sense can an interaction and traveling down a path in a hyper-network be expected to lead to solutions or to insights. It may be equivalent to the necessary revival of the social-pragmatic dimension of mathematics (Davis & Hersh, 1986, pp. 154, 163, 197-199, 252; Rosen, 1985b; Rota, 1986b, pp. 249ff.).

In hypertext terms this seem to have been recognized in recent approaches that are akin to social systems-hypergames that expose systems people of all classes to some of the complexities of social systems design options (Fischer, et al., 1989; Forsgren, 1988; 1989). It is apparent, however, that technology obfuscates many subtle conceptual system-questions as when a "kitchen design" (Fischer, et al., 1989) is compared with more social alternative conceptions of dishing (Ivanov, 1986, pp.64-67) or of the design of a chinese restaurant kitchen (Davis & Hersh, 1986, pp. 127-131). In another more sophisticated example, a metaphor was constructed based on the creation of game-like learning environments, nested multiboard games of chess which provide possibilities, pointing ways out of the international arms stalemate by supporting current arms negotiations (McWhinney, Greening, & Mitroff, 1988). Our proposed research will also attempt, in the context of metaphorical approaches, to encompass aesthetical and ethical dimensions by extending the concept of games with respect to the mythological dimension that is already well exploited in the technology for computer games (Mitroff, 1983; Mitroff, 1984).

In an analog way, it is possible that combinatorial algebraic topology (Rota, 1986a) or topological matrix algebra lies at the heart of the matrix analysis of information networks and information systems architecture (Langefors, 1973; Langefors & Sundgren, 1975), including structure of programs (Davis & Hersh, 1986, p.179, 182). It probably suffers of the same deficiency as graph theory for dealing with the social dimensions of formalism. To the extent that such graphical system theories as included in several surveys (Couger & Knapp, 1974, as an early overview of the field; Olle, et al., 1986; Olle, et al., 1983; Olle, et al., 1982) and associated visual methods (Lundeberg & Andersen, 1974) are not developed into a social systems theory as suggested above, it is to be expected that human science aspects, e.g. sociological and psychological, will give rise to the need of supplementary programming-support research and activities that are difficult to classify. Available in-depth critique of certain visual graph methods (Bødker & Hammerskov, 1982) point clearly in that direction.

These rather underterminate program-support activities, a kind of empirically based interactivity ergonomics that sometimes is labeled man-machine interaction or user interface, lately in close contact with so called cognitive science, may often work as a kind of ragbag for taking care of the symptoms or consequences of "dehumanized" formal science as implicit in inadequate software. By now a considerable body of common sense knowledge that seemingly is philosophically neutral has been systematized in various ways and is well represented by published literature (Johansen, Rijnsdorp, & Sage, 1983; Margulies & Zemanek, 1983; Newman & Sproull, 1979; Shneiderman, 1980; 1987). It can be regarded as the computer age heir of early systematizations of data on "human performance" and "psychological principles of systems development" (Fitts & Poser, 1973; Gagné, 1962).

Close to matters of man-machine interface and interaction stand educational aspects of software research. Such aspects are not intended here in the sense of their latest fashionable expression of computer aided instruction and "courseware" (Schwartz, 1986c) nor in the sense of supposed learning capabilities of e.g."connection machines" (Hillis, 1986). What we have in mind are the attempts that have been made to give an unitary image of the software field, with an outlook towards both past and future, in order to "educate" researchers and students or in order to give the view of some particular development (Abelson & Sussman, 1983; Balzer, Cheatham, & Green, 1982; Baron, 1986; Bubenko, 1982; Dijkstra, 1966; Dijkstra, 1972; Dijkstra, 1976; Gorn, 1963; Harel, 1987; Lunell, 1981; Mayoh, 1984; Minsky, 1970; Naur, 1968; Newell, Perlis, & Simon, 1967; Newell & Simon, 1981; Owen, 1983; Pratt, 1983; Ritchie, 1984).

Some of these attempts are elegantly concise (Owen, 1983), but very few, if any, of them acknowledge the need for incorporating a social or economic dimensions in the formation of such unitary views. A hint on how such a dimensions could appear in this kind of studies was given in one case when analyzing the success of the UNIX operating system (Ritchie, 1984). In another case the pragmatic aspects in selecting a language, compiler, and support environment were considered thanks to the use of an operation research approach (Anderson & Shumate, 1982). In a third case there was an attempt to incorporate the concept of "capital" while discussing the software technology's trend towards "resuability" that is considered as a "general engineering principle" which on the basis of capital formation provides a "stock" of programming "tools" (Wegner, 1982).

An interesting research object would be to advance the understanding of social dimensions of software development by means of a critical analysis of leftist treatments of software tools (Ehn, 1988). In that context it could be useful to take into consideration some of the more encompassing approaches to the social aspects of the technological dimension in order to complete their deficient integration with software technical issues, e.g. in the context of the relation between technology and engineering (Hood, 1982; Sinclair & Tilson, 1982), between programming and organizational development or work environment (Kraft, 1977; Kraft, 1979), between programming and user interface design beyond the limited framework of cognitive science (Johansen, et al., 1983; Margulies & Zemanek, 1983), or between user interface design and work environment (Ehn, 1988).

An important research strategy for dealing with these more encompassing issues will be "the case study". In a way that is similar to the previously mentioned study of the reasons for the success of UNIX, the fortune of SIMULA, of APL (Falkoff & Iverson, 1981) and of influential AI-expert systems like DENDRAL (Churchman, 1971, chap.4; Churchman & Buchanan, 1969; McCorduck, 1979) and MYCIN (Shortliffe, 1976) could also be studied. Even the highly technical-formal field of compiler-compilers that is seemingly immune to social factor may show, on closer analysis, the role of human and subjective factors. The following example will may suffice. In discussing research ideals for systems which attempt to automate the design of code-generators or compilers, i.e. "code generator writing systems" CGWS (Lunell, 1983, p.264), it is suggested that the appropriateness of automating a supporting CGWS (turning it into an *automatic* code generator generator, ACGG) depends upon whether the *user* of the ACGG is the same person as the author of the language of target machine: "The language designer using the ACGG, [in order] to be able to execute programs in the language he is designing, needs diagnostics related to source language, and the opposite is true about the machine designer [who needs them in terms of target machine description]. The person using the systems just to achieve a new language-machine combination is helpless regardless of the type of diagnostics." Our contention is that his may, in last analysis, be regarded as a complex question of what does it mean to be an "expert" or "support", and to work from *within* a given system versus to view the system from the *outside.* It is also a question of what is to be meant by *input* to a system, a matter which is slightly hinted at in Lunell, and is developed elsewhere (Churchman, 1971). It is also an example of how the subjective and social dimensions can and should enter discussions about technical and formal software matters. Up to now this has not been usual (Aho & Ullman, 1979, seems to be representative in this respect).

An alternative way to considering human science aspects of software research, besides more philosophical and "speculative" approaches, e.g. in the spirit of A. Korzybski's "general semantics" (Korzybski, 1948; Neumann, 1979; Neumann, 1982), is to "psychologize" the taxonomies for programming languages mentioned earlier trying to avoid falling into the trap of using the psychologizing as a ragbag for problems that are not understood properly. It has been suggested that e.g. object-oriented languages would be adequate for applications to physical manufacturing of discrete objects, while production or process-oriented language structures would be adequate for administrative processes according to deductive rules. This idea could be expanded by imagining that a particular type of languages would be adequate for matching the interface towards users with a particular "cognitive style". The issue of cognitive styles is well outlined in certain literature (Benbasat & Taylor, 1978; Gough & Woodworth, 1960; McKenney & Keen, 1974) but it will be developed elsewhere. Here it will suffice to mention that this psychologizing may create insights that recall those related to the "limit values" in physics, in the sense that it will focus the user interface as a limit where there is a confluence of three aspects of program design. They are those which are vaguely labeled as knowledge representation, type of problem or domain, and mental models of users. It is, for instance, sometimes stated that the knowledge representation used in the machine should "mirror" the knowledge representation or mental model of the end user. The users' interactions with the system are then supposed to be performed via an interface that explicitly mirrors their mental model of the problem domain (R. Whitaker, pers. communication). All this is conceptualized in a particular terminology (Steltzner & Williams, 1988) that can be regarded as a technologically conditioned updating and widening of earlier approaches (Sheil, 1981), without the benefit of late socio-political approaches to the concept of user interface (Ehn, 1988; Nilsson, 1987, presents a survey of the user interface issue).

It should be recalled that in ultimate analysis the problem of knowledge representation is epistemologically grounded in Kant's philosophy (Churchman, 1971, chap.6) that was also the opening door for relevant currents of modern psychology to be considered elsewhere in the context of logic, mathematics and cultural criticism. The shipwreck of this issue in the modern software debate may correlate with the difficulties raised by the Kantian doctrine in general, and denounced in the romantic reaction in particular (Steiner, 1886/19XX; Steiner, 1937/1982). The very same difficulties may be the basis and background for the challenges that have been directed lately concerning the soundness of the development of modern applied mathematics, including its embodyment in computers (Barrett, 1987; Davis & Hersh, 1986).

With regard to unclarities of modern mathematics it is to be noted that to our knowledge, nowhere has the concept of function, so often invoked in computer science, been questioned so deeply as it has been in the field of mathematics itself. In computer science it is, for instance, claimed that "essentially every software activity is the evaluation of a function *f*; ...however, the nature of *f* can differ from case to case"; functions can be *rule-based* as in conventional programs, and *look-up based* as for data-base queries, or through an "orthogonal partitioning" they can be classified as *stable* and *dynamic,* the present development pointing toward *rule-based dynamic functions* (Berztiss, 1988). There are still more ambitious and questionable approaches to functions in computer science (Doyle, 1982). While earlier pioneers of computer science (Gorn, 1964; Korfhage, 1964) still kept a close contact with the formal disciplines that could clarify such deep and easily misused concepts, today's practitioners do not seem to have the opportunity to partake of the illuminating work and criticism that is going from within these disciplines. For instance, what is a computer program and a particular computation when they are conceptualized in the language of mathematics? (Chaitin, 1974.) What may be wrong with the modern concept of mathematical-logical function? (Davis & Hersh, 1986, pp. 190, 194-195; Rota, 1986c, p.177ff.; Rychlak, 1977, pp. 50, 256). What relations does it have with other concepts of function? (Ackoff & Emery, 1972, chap. 1-2, 9-10, 15.; Verene, 1982. p. 287) Further, what may be wrong with the modern functional treatment of the concept of time, that has caused so much trouble in data base theory? (Davis & Hersh, 1986, pp. 193, 199-200; Rota, 1986d, pp. 168, 171; Rychlak, 1977, p.288).

A basic hypothesis for our proposed research is that close contact with formal sciences including their critical undercurrents will be fruitful for understanding the causes of many present difficulties and for improved design of software. That is also one of the reason why in our references we have sought to relate mostly to formal scientists, mainly professional mathematicians, who have a grasp of the foundational problems of their discipline. More details about this question will be introduced in the context of mathematics and logic proper. Another basic hypothesis that guarantees our keeping the contact between programming theory and organization theory is the interpretation of programming in terms of inquiring systems. The final contention will be, at least at the level of basic research, that the obtention of programming languages "that make it possible to use small independent fragments to define complex processes" (Langefors, 1973; Schwartz, 1986a, p. 189), i.e. the realization of software *systems,* is fundamentally and ultimately a question of inquiring systems and organization (Churchman, 1971). It is an insight that seems to emerge quite clearly from some recent treatments of the software-hardware issue in terms of applications of social systems theory (Sachs & Broholm, 1989; Swanson, 1989).

This puts into question the definition or delimitation of the concept of software itself and it will also be the bridge between our project for research on software and the related project for research on economic and organizational analysis of business systems.

References

Abelson, H., & Sussman, G. J. (1983). Structure and interpretation of computer programs. No. Cambridge: MIT, Dept. of Electrical Engineering and Computer Science).

Ackoff, R. L., & Emery, F. E. (1972). On purposeful systems: An interdisciplinary analysis of individual and social behavior as a system of purposeful events. Chicago: Aldine-Atherton.

Aho, W., & Ullman, J. D. (1979). Principles of compiler design. Reading, Mass.: Addison-Wesley.

Anderson, G. E., & Shumate, K. C. (1982). Selecting a programming language, compiler, and support environment: method and example. (August), 29-36.

Annerstedt, J., Forssberg, L., Henriksson, S., & Nilsson, K. (1970). Datorer och politik. Studier i en ny tekniks politiska effekter på det svenska samhället. Staffanstorp: Zenith & Bo Cavefors.

Augarten, S. N. Y., 1984. (1984). Bit by bit: An illustrated history of computers. New York: Ticknor and Fields.

Backus, J. W. (1960). The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM Conference. In UNESCO (Eds.), Information processing. Proceedings of the Int. Conf. on Information Processing in Paris, 15-20 June 1959 (pp. 125-132). London-München: Butterworths-Oldenbourg.

Balzer, R., Cheatham, T. E., & Green, C. (1982). Software technology in the 1990's: Using a new paradigm. Computer, 16(11, November), 39-45.

Bar-Hillel, Y. (1973). Language and information: Selected essays on their theory and application. Reading, Mass: Addison-Wesley.

Bar-Hillel, Y., & Carnap, R. (1953). Semantic information processing. British J. for the Philosophy of Science, 4, 147-157.

Baron, N. S. (1986). Computer languages: A guide for the perplexed. New York: Anchor Press.

Barrett, W. (1987). Death of the soul: From Descartes to the computer. Oxford: Oxford University Press.

Bauer, F. L., & Samelson, K. (1960). The problem of a common language, especially for scientific numeral work. In UNESCO (Eds.), Information processing. Proceedings of the Int. Conf. on Information Processing in Paris, 15-20 June 1959 (pp. 120-125). London-München: Butterworths-Oldenbourg.

Belnap, N., & Steele, T. B. (1976). Logic of questions and answers. New Haven: Yale University Press.

Ben-Ari, M. (1982). Principles of concurrent programming. Englewood Cliffs: Prentice-Hall International.

Benbasat, I., & Taylor, R. N. (1978). The impact of cognitive styles on information system design. MIS Quarterly, 2(2), 43-54.

Berild, S. (1981). A data base design tool and its use in a large data base application No. SYSLAB report No.2). The Royal Institute of Technology and University of Stockholm, Dept. of Information Processing and Computer Science.

Berztiss, A. T. (1988). A course on artificial intelligence for software engineers No. SYSLAB report No.58). The Royal Institute of Technology and University of Stockholm, Dept. of Information Processing and Computer Science.

Björnestedt, A., & Britts, S. (1988). AVANCE: An object management system No. SYSLAB report No. 60). The Royal Institute of Technology and University of Stockholm, Dept. of Information Processing and Computer Science.

Bowers, D. M. (1977). The rough road to today's technology. Datamation, 23(9), 69-74.

Brouwer, L. E. J. (1905). Leven, Kunst en Mystiek [Life, art and mysticism]. In Delft: Waltman.

Bubenko, J. (1982). Systemutveckling 1981: En granskning av teori- och metodutvecklingen samt systemutvecklarens roll. Data(February 1st), 41-45.

Bubenko, J. (1983). Information and data modeling: State of the art and research directions No. SYSLAB report No.20). The Royal Institute of Technology and University of Stockholm, Dept. of Information Processing and Computer Science.

Bubenko, J. (1988). Problems and unclear issues with hierarchical business activity and data flow modeling No. SYSLAB report WP-134, rev. June 1988). The Royal Institute of Technology and University of Stockholm, Dept. of Computer and Systems Science.

Buchanan, B. G. (1986). Expert systems: Working systems and the research literature. Expert Systems, 3(1).

Bäckström, C. (1988). Reasoning about interdependent actions No. Licentiate in Engineering thesis No.139, ISBN 91-7870-358-1, ISSN 0280-7971). University of Linköping, Dept of Computer and Information Science.

Bødker, S., & Hammerskov, J. (1982). Grafisk systembeskrivelse: Analyse af ISAC V- og I-grafværktøjer på baggrund av et eksempel og en sammenligning med andre grafiske systembeskrivelsesværktøjer No. DAIMI IR-33 and IR-35. ISSN 0106-9969). University of Aarhus, Matematisk Institut - Datalogisk afdelning.

Chaitin, G. J. (1974). Information-theoretic limitations of formal systems. J. of the ACM, 21(3, July), 403-424.

Checkland, P. B. (1981). Systems thinking, systems practice. New York: Wiley.

Checkland, P. B. (1988). Soft systems methodology: An overview. J. of Applied Systems Analysis, 15, 27-30.

Churchman, C. W. (1948). Theory of experimental inference. New York: Macmillan.

Churchman, C. W. (1961). Prediction and optimal decision: Philosophical issues of a science of values. Englewood Cliffs: Prentice-Hall.

Churchman, C. W. (1971). The design of inquiring systems: Basic principles of systems and organization. New York: Basic Books.

Churchman, C. W. (1979). The systems approach and its enemies. New York: Basic Books.

Churchman, C. W., & Buchanan, B. G. (1969). On the design of inductive systems: Some philosophical problems. Br. J. Phil. Sci., 20, 311-323.

Cohen, P. R., & Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. Cognitive Science, 3(3), 177-212.

Computer science curriculum (1964). Comm. of the ACM, 7(4, April), 205-230.

Couger, J. D., & Knapp, R. W. (1974). Systems analysis techniques. New York: Wiley.

Davis, P. J., & Hersh, R. (1986). Descartes' dream: The world according to mathematics. New York and London: Harcourt Brace Jovanovich, and Penguin Books.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. CACM, 32(1, January), 9 ff.

Dijkstra, E. W. (1966). Programming considered as a human activity. In Proc. of the 3rd IFIP Congress, New York, 1965 (pp. 213-217). Washington & London.

Dijkstra, E. W. (1972). The humble programmer. Communications of the ACM, 15(10, October), 859-866.

Dijkstra, E. W. (1976). A discipline of programming. Englewood Cliffs: Prentice-Hall.

Doyle, J. (1982). The foundations of psychology: A logico-computational inquiry into the concept of mind No. Report CMU-CS-82-149). Pittsburgh: Carnegie Mellon University, Dept. of Computer Science.

Dunn, E. S., Jr. (1974). Social information processing and statistical systems: Change and reform. New York: Wiley.

Ehn, P. (1988). Work-oriented design of computer artifacts. (Doctoral diss.). Umeå-Stockholm: University of Umeå, Arbetslivscentrum and Almqvist & Wiksell International.

Elbourn, R. D., & Ware, W. H. (1962). The evolution of concepts and languages of computing. Proc. of the IRE, 50(5, May), 1059-1066.

Encyclopedia of computer science and engineering (1983). (A. Ralston, Ed.; 2nd ed.). New York: Van Nostrand Reinhold.

Evans, C. (1981). The making of the micro. New York: Van Nostrand Reinhold.

Falkoff, A. D., & Iverson, K. E. (1981). A source book in APL. Palo Alto: APL Press.

Fischer, G., Morch, A., & McCall, R. (1989). Design environments for constructive and argumentative design. In K. Bice & C. Lewis (Eds.), The ACM Conference on Human Factors in Computing Systems (CHI '89): 'Wings for the mind'. Austin, Texas, April 30-May 4 1989

Fisher, W. R. (1987). Technical logic, rhetorical logic, and narrative rationality. Argumentation: An Int. J. on Reasoning (Dordrecht), 1(1), 3-21.

Fitts, P. M., & Poser, M. I. (1973). Human performance. London: Prentice-Hall International.

Forsgren, O. (1988). Samskapande datortillämpningar [Constructive computer applications] No. Doctoral diss., Report UMADP-RRIPCS-3.88). University of Umeå, Inst. of Information Processing.

Forsgren, O. (1989). The first "co": A prototype of a learning co-constructor. In Proc. of the ISSS Int. Society for the Systems Sciences, 33rd Annual Conference, Edinburgh, Scotland, 2-7 July 1989. Vol. 1 (pp. 92-97).

Forsythe, G. (1967). A university's educational program in computer science. Comm. of the ACM, 10(1, Jan.), 3-11.

Forsythe, G. E. (1970). Educational implications of the computer revolution. In Z. W. Pylyshyn (Eds.), Perspectives on the computer revolution (pp. 378-389). Englewood Cliffs: Prentice-Hall.

Freedle, R. O. (1975). Dialogue and inquiring systems. Human Dev, 18, 97-118.

Gagné, R. M., (Ed.) (1962). Psychological principles in system development. New York: Holt, Rinehart and Winston.

Gaines, B. R., & Shaw, M. L. G. (1984). The art of computer conversation. Englewood Cliffs: Prentice-Hall.

Gharajedaghi, J. (1986). Development of practitioners: Incompatibility of educational media and messages No. INTERACT - The Institute for Interactive Management, 3440 Market St, Philadelphia PA 19104.

Goldkuhl, G., & Lyytinen, K. (1982). A language action view on information systems. In Proc. of the 3rd IFIP Int. Conf. on Information Systems, Ann-Arbor, Mich, 1982

Gorn, S. (1960). On common symbolic language for computers. In Information processing. In UNESCO (Eds.), Information processing. Proc. of the international conference on information processing in Paris, 15-20 June 1959 (pp. 117-120). London and München: Butterworths and Oldenbourg.

Gorn, S. (1963). The computer and information sciences: A basic discipline. SIAM Review, 5, 150-155.

Gorn, S. (1964). Mechanical languages: A course specification. Comm. of the ACM, 7(4, april), 219-222.

Gough, H. G., & Woodworth, D. W. (1960). Stylistic variations among professional research scientists. Journal of Psychology, 49, 87-98.

Grindley, C. B. B. (1966). Systematics: A non-programming language for designing and specifying commercial systems for computers. The Computer Journal, 9(2), 124-128.

Grosch, H. R. J. (1977). The way it was: 1957. Datamation, 23(9, September), 75-79.

Guthrie, E. R. (1942). The principle of associative learning. In F. P. Clarke & M. C. Nahm (Eds.), Philosophical essays in honor of Edgar Arthur Singer, Jr. (pp. 100-114). Philadelphia and London: University of Pennsylvania Press and Humphrey Milford Oxford University Press.

Harel, D. (1987). Algorithmics: The spirit of computing. Reading, Mass.: Addison-Wesley.

Harland, D. M. (1984). Polymorphic programming languages. New York: Wiley.

Henson, M. C. (1987). Elements of functional languages. Oxford: Blackwell Scientific Publications.

Hester, S. D., Parnas, D. L., & Utter, D. F. (1981). Using documentation as a software design medium. The Bell Systems Technical J., 60(8, October), 1941-1977.

Hillis, W. D. (1986). The connection machine. Cambridge: The MIT Press.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. Comm. of the ACM, 12(10, Oct.), 576-583.

Hodges, A. (1983). Alan Turing: The enigma. London: Simon & Schuster.

Holbaek-Hanssen, E., Håndlykken, P., & Nygaard, K. (1985). System description and the DELTA language No. DELTA project report No.4, publication No.523). Oslo: Norwegian Computing Center - Norsk Regnesentral, 1985.

Hood, W. F. (1982). Dewey and technology: A phenomenological approach. Research in Philosophy and Technology, 5, 189-207.

Humphreys, P., & Berkeley, D. (1988). Conceptual model building: Capturing and representing purposeful activity and knowledge in the organisation No. Paper presented at the IFIP WG 8.3 Working Conference on Organizational Decision Support Systems, Lake Como, Italy, 20-22 June, 1988). London School of Economics and Political Science, Dept. of Social Psychology.

Information systems-curriculum recommendations of the 80's (1982). Undergraduate and graduate programs; A report of the ACM Curriculum Committee on Information Systems. Comm. of the ACM, 24(11, Nov.), 781-805.

Ivanov, K. (1972). Quality-control of information: On the concept of accuracy of information in data banks and in management information systems No. Doctoral diss.). The University of Stockholm and The Royal Institute of Technology.

Ivanov, K. (1976). Statistiska informationssystem: Framväxten av en svensk skola om data och information och dess förhållande till en kunskap om sociala informationsprocesser No. Research report No.1976:8, ISSN 0347-2108). University of Stockholm, Dept. of Statistics.

Ivanov, K. (1979). Datorbaserad social kommunikation: Inskränkt till ett högnivåspråk för programmering av motsägelselösa samhällsbeskrivningar [Computer-based social communication: Reduced to a high-level language for programming of unambiguous descriptions of society]. Statistisk Tidskrift(3), 173-187.

Ivanov, K. (1983). Några reflektioner över framtida datautbildning och forskning. In Kunskap för framtiden (pp. 79-91). Stockholm: Liber Utbildningsförlaget.

Ivanov, K. (1984). Systemutveckling och ADB-ämnets utveckling. In G. Goldkuhl (Eds.), Systemutveckling, av vem, för vem och hur? (Report No. K4/84) Stockholm: Arbetarskyddsfonden.

Ivanov, K. (1986). Systemutveckling och rättssäkerhet : Om statsförvaltningens datorisering och de långsiktiga konsekvenserna för enskilda och företag [Systems development and rule of law]. Stockholm: SAF:s Förlag.

Ivanov, K. (1988). Expert-support systems: The new technology and the old knowledge. Systems Research, 5(2), 293-100.

Janlert, L. E. (1988). Människa-dator interaktion No. Umeå, Sweden: University of Umeå, Inst. of Information Processing.

Jobs, S. (1989). Vandringen är målet. Stockholm: Columna.

Johansen, G., Rijnsdorp, J. E., & Sage, A. P. (1983). Human system interface concerns in support system design. Automatica, 19(6), 595-603.

Jönsson, A. (1987). Naturligt språk för användardialog och databasförfrågningar No. Report LiTH-IDA-R-87-25, Industriserien). University of Linköping, Dept. of Computer and Information Science.

Kac, M., Rota, G. C., & Schwartz, J. T. (1986). Discrete thoughts: Essays on mathematics, science, and philosophy. Boston: Birkhäuser.

Katz, C. S., Doria, F. A., & Lima, L. C. (1975). Dicionário básico de comunicação (2nd ed.). Rio de Janeiro: Paz e Terra.

Kay, A. (1984). Computer software. Scient. Am., 251(3), 53-59.

Kidder, T. (1982). The soul of a new machine. Harmondsworth, Middlesex: Penguin.

Kleene, S. C. (1950). Introduction to metamathematics. Princeton: Van Nostrand.

Klir, G. J., & Lowen, W. (1989). The world of mathematics mirrors the organization of the mind: Part I, Mathematics and systems science; Part II, Mathematics, the mind and the brain No. Research report). New York State University at Binghamton, Thomas J. Watson School, Dept. of Systems Science.

Korfhage, R. R. (1964). Logic of the computer sciences. Comm. of the ACM, 7(4, April), 216-218.

Korzybski, A. (1948). Science and sanity: An introduction to non-Aristotelian systems and general semantics (3rd ed.). Lakeville, Conn.: The Institute of General Semantics.
Kowalski, R. (1979). Algorithm = Logic + Control. CACM, 7, 424-436.
Kraft, P. (1977). Programmers and managers: The routinization of computer programming in the United States. New York: Springer Verlag.
Kraft, P. (1979). The industrialization of computer programming: From programming to "software production". In A. Zimbalist (Eds.), Case studies on the labour process New York: Monthly Review Press.
Lammers, S. (1988). De stora programmerarna. Stockholm: Columna.
Langefors, B. (1973). Theoretical analysis of information systems. Philapelphia: Auerbach.
Langefors, B., & Sundgren, B. (1975). Information systems architecture. New York: Petrocelli/Charter.
Levén, P., & Nordström, T. (1989). Socially responsive systems No. Presented at the conference Support, society and culture: Mutual uses of cybernetics and science, The Int. Federation for Cybernetics, Amsterdam, March 27-31, 1989. Dept. of Andragology, OOC–program, Grote Bickersstraat 72, 1013 KS Amsterdam).
Levin, J. A., & Moore, J. A. (1988). Dialogue games: Metacommunication structures for natural language interaction. In Readings in distributed artificial intelligence (pp. 385-397). San Mateo, Calif.: Morgan Kaufmann.
Lundeberg, M., & Andersen, E. S. (1974). Systemering - Informationsanalys. Lund: Studentlitteratur.
Lunell, H. (1981). Tre skisser om datalogi som vetenskap No. Report LiTH-MAT-R-81-16, ISSN 0348-2960). University of Linköping, Dept. of Computer and Information Science.
Lunell, H. (1983). Code generator writing systems No. Doctoral diss. No.94). University of Linköping, Dept. of Computer and Information Science.
Lyytinen, K. (1986). Information systems development as social action: framework and critical implications (Doctoral diss. No. Doctoral diss.). University of Jyväskylä, Finland, Dept. of computer science.
Lyytinen, K. (1988a). Expectation failure concept and system analyst's view of information system failures: Results of an exploratory study. Information and Management, 14, 45-56.
Lyytinen, K. (1988b). Stakeholders, information system failures and soft systems methodology: An assessment. J. of Applied Systems Analysis, 15, 61-81.
Macmurray, J., Ewing, A. C., & Franks, O. S. (1938). Symposium: What is action? In Action, perception and measurement (pp. 69-120). London: Harrison and Sons.
Margulies, F., & Zemanek, H. (1983). Man's role in man-machine systems. Automatica, 19(6), 677-683.
Mayoh, B. H. (1984). Comparative semantics of programming No. DAIMI PB-173, ISSN 0105-8517). Aarhus University, Dept of Computer Science.
McCorduck, P. (1979). Machines who think: A personal inquiry into the history and prospects of artificial intelligence. San Francisco: Freeman.
McKenney, J. L., & Keen, P. G. W. (1974). How managers' minds work. Harvard Business Review(May-June), 79-90.
McWhinney, W., Greening, T., & Mitroff, I. (1988). Four levels of nuclear reality No. Unpublished manuscript). University of Southern California, Graduate School of Business.

Mendner, J. H. (1976). Teknologisk utveckling i den kapitalistiska arbetsprocessen. Copenhagen: Kurasje.

Minsky, M. (1970). Form and content in computer science. J. of the ACM, 17(2, April).

Mitroff, I. I. (1983). Archetypal social systems analysis: On the deeper structure of human systems. Academy of Management Review, 8(3), 387-397.

Mitroff, I. I. (1984). The invasion of the mind: A worst possible scenario for the office of the future. Office: Technology and People(2), 79-86.

Morgenstern, O. (1963). On the accuracy of economic observations (2nd ed.). Princeton: Princeton University Press.

Naur, P. (1968). "Datalogy": The science of data and data processes. In Proc. of the IFIP Congress, 1968 (pp. 1383-1387).

Neumann, P. G. (1979). A note on the psychology of abstraction. ACM-SIGSOFT Software Engineering Notes, 4(1, Jan.), p. 21.

Neumann, P. G. (1982). Psychosocial implications of computer software development and use: Zen and the art of computing. ACM-SIGSOFT Softaware Engineering Notes, 7(2, April), 3-11.

Newell, A., Perlis, A., & Simon, H. (1967). Computer science. Science, 157(22 September).

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem solving program. In UNESCO (Eds.), Information processing. Proceedings of the UNESCO Int. Conf. on Information Processing in Paris, 15-20 June 1959 (pp. 256-264). London and München: Butterworths and Oldenbourg.

Newell, A., & Simon, H. A. (1981). Computer science as empirical inquiry. In J. Haugeland (Eds.), Mind design: Philosophy, psychology, artificial intelligence Montgomery, Vermont: Bradford Books.

Newman, W., & Sproull, R. (1979). Principles of interactive computer graphics. New York: McGraw-Hill International.

Nilsson, K. (1987). Project description: Design of interactive information systems No. Report UMADP-RRIPCS-5.87, ISSN 0282-0579). Inst. for Information Processing, University of Umeå, Inst. of Information Processing.

Nygaard, K. (1988). O4 - Object oriented office organization: A proposal for the ESPRIT technical integration project "Integrated application support system" No. Unpublished lecture notes). The Norwegian Computing Center, Oslo.

Olle, T. W., H.J., S., & Verrijn-Stuart, A. A. (1986). Information system design methodologies: Improving the practice. Amsterdam: North Holland.

Olle, T. W., Sol, H. G., & Tully, C. J., (Eds.) (1983). Information systems methodologies: A feature analysis. Amsterdam: North Holland.

Olle, T. W., Sol, H. G., & Verrijn-Stuart, A. A., (Eds.) (1982). Information systems design methodologies: A comparative review. Amsterdam: North Holland.

Orchard-Hays, W. (1961). The evolution of programming systems. Proc. of the IRE, 49(1, January), 283-295.

Orci, I. P. (1983). Contributions to automatic programming theory: A study in knowledge-based computing No. Doctoral diss., report TRITA-IBADB-1105, ISBN 91-85212-97-0). Royal Inst. of Technology and Univ. of Stockholm, Dept. of Information Processing and Computer Science.

Owen, K. (1983). The art and science of programming: IFIP's community of experts tackle key problem areas. Algol Bulletin (ISSN 0084-6198)(December 1983, No.50).

Papert, S. (1980). Mind storms: Children, computers, and powerful ideas. New York: Basic Books.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Comm. of the ACM, 15(12, Dec.), 1053-1058.

Parnas, D. L. (1976). On the design and development of program families. IEEE Trans. Software Engineering, SE-2(March), 1-9.

Parnas, D. L. (1985). Software aspects of strategic defense systems. American Scientist, 73(5, Sept-Oct), 432-440.

Parnas, D. L. (1989a). Education for computing professionals No. Technical Report 89-247, ISSN 0836-0227). Dept. of Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6.

Parnas, D. L. (1989b). A mathematical basis for computer assisted software engineering No. Announcement in IDA-Kuriren of a guest lecture held on January 17th, 1989). Dept. of Computer and Information Science, University of Linköping.

Parnas, D. L., & Clements, P. C. (1986). A rational design process: How and why to fake it. IEEE Trans.on Software Engineering, SE-12(2, Feb.), 251-257.

Parnas, D. L., Clements, P. C., & Weiss, D. (1984). The modular structure of complex systems. In Proc. of the 7th Int. Conference on Software Engineering, Orlando, Florida, March 1984 (pp. 408-417). New York: IEEE Press.

Parnas, D. L., van Schouwen, A. J., & Kwan, S. P. (1988). Evaluation standards for safety critical software No. Technical report 88-220, ISSN 0836-0227). Kingston, Ontario: Queen's University, Dept. of Computing & Information Science.

Pask, G. (1976). Conversation theory: Applications in education and epistemology. Amsterdam: Elsevier.

Pask, G., & Gregory, D. (1987). Conversational systems. In J. Zeidner (Eds.), Human productivity enhancement - Vol.2 (pp. 204-235). New York: Praeger.

Pirsig, R. (1974). Zen and the art of motorcycle maintenance. New York: Bantam Books.

Poyen, J., & Vauquois, B. (1960). Suggestions for a universal language. In Information processing. Proceedings of the UNESCO Int. Conf. on Information Processing in Paris, 15-20 June 1959 (pp. 132-138). London and München: Buttertworths and Oldenbourg.

Pratt, V. (1983). Five paradigm shifts in programming language design and their realization in Viron, a dataflow programming environment. In Conference ecord on the tenth annual ACM symposium on principles of programming languages, Austin, Texas, Jan. 1983 (pp. 1-10).

Pylyshyn, Z. W., (Ed.) (1970). Perspectives on the computer revolution. Englewood Cliffs: Prentice-Hall.

Quiniou, J. C. (1971). Marxisme et informatique. Paris: Éditions Sociales.

Randell, B., (Ed.) (1973). The origins of digital computers: Selected papers. Berlin: Springer Verlag.

Reid, C. (1970). Hilbert. New York: Springer Verlag.

Rheingold, H. (1985). Tools for thought: The people and the ideas behind the next computer revolution. New York: Simon & Schuster.

Richards, H., Jr. (1985). Applicative programming. Systems Research, 2(4), 299-306.

Ritchie, D. M. (1984). Reflections on software research (Turing award lecture). Comm. of the ACM, 27(8, Aug.), 758-763.

Robson, D. (1981). Object-oriented software systems. Byte, 6(8), 74-86.

Roosen-Runge, P. H. (1989). Programming languages considered harmful: A sceptical look at the central dogma of computer science No. Draft notes of lecture given on June 7, 1989, at the University of Umeå, Inst. of Information Processing). Dept. of Computer Science, York University, 4700 Keele St., North York, Ontario M3J IP3, Canada.

Rosen, R. (1985a). Organisms as causal systems which are not mechanisms: An essay into the nature of complexity. In R. Rosen (Eds.), Theoretical biology and complexity New York: Academic Press.

Rosen, R. (1985b). The physics of complexity. Systems Research, 2(2), 171-175.

Rota, G. C. (1986a). Combinatorics. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 49-62). Boston: Birkhäuser.

Rota, G. C. (1986b). Heidegger. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 247-252). Boston: Birkhäuser.

Rota, G. C. (1986c). Husserl. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 175-181). Boston: Birkhäuser.

Rota, G. C. (1986d). Husserl and the reform of logic. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 167-173). Boston: Birkhäuser.

Rota, G. C. (1986e). Mathematics: trends. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 77-103). Boston: Birkhäuser.

Rychlak, J. F. (1977). The psychology of rigorous humanism. New York: Wiley.

Sachs, W., & Broholm, P. (1989). Hypertrophy in the micro-computer revolution. In C. W. Churchman (Eds.), The well-being of organizations (pp. 171-177). Salinas, Calif.: Intersystems.

Sandewall, E., & Rönnquist, R. (1986). A representation of action structures No. Research report LiTH-IDA-R-86-13, also in Proc. of the AAAI-86, Philadephia, 1986). University of Linköping, Dept of Computer and Information Science.

Schneider, W. (1980). Some aspects on formalization techniques in medicine and health care. In D. Lindberg & Z. Kaihara (Eds.), Proc. of MedInfo '80 (pp. 1276-1280). Amsterdam: North-Holland.

Schwartz, J. T. (1962). The pernicious influence of mathematics on science. In E. Nagel, P. Suppes, & A. Tarski (Eds.), Logic, methodology and philosophy of science (Proc. of the 1960 International Congress) (pp. 356-360).

Schwartz, J. T. (1986a). Artificial intelligence. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 183-190). Boston: Birkhäuser.

Schwartz, J. T. (1986b). Computer science. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 63-76). Boston: Birkhäuser.

Schwartz, J. T. (1986c). Computer-aided instruction. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 207-230). Boston: Birkhäuser.

Schwartz, J. T. (1986d). The future of computer science. In M. Kac, G. C. Rota, & J. Schwartz T. (Eds.), Discrete thoughts: Essays on mathematics, science, and philosophy (pp. 105-116). Boston: Birkhäuser.

Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. AIEE Transactions, 57, 713-723.

Sheil, B. A. (1981). The psychological study of programming. Computing Surveys, 13(1, March), 101-120.

Shneiderman, B. (1980). Software psychology. Cambridge: Winthrop Publishers.

Shneiderman, B. (1987). Designing the user interface. Reading, Mass.: Addison-Wesley.

Shortliffe, E. H. (1976). Computer based consultation: MYCIN. New York: American Elsevier.

Sinclair, G., & Tilson, W. V. (1982). The relationship of technology to engineering. Research in Philosophy and Technology, 5, 87-97.

Singer, E. A., Jr. (1959). Experience and reflection. Philadelphia: University of Pennsylvania Press.

Solomonides, T., & Levidow, L., (Eds.) (1985). Compulsive technology: Computers as culture. London

Free Association Books:

Sowa, J. F. (1984). Conceptual structures: Information processing in mind and machine. Reading: Addison-Wesley.

Steiner, R. (1886/19XX). A theory of knowledge implicit in Goethe's world conception. London: Rudolf Steiner Press.

Steiner, R. (1937/1982). Nascita e sviluppo storico della scienza (Schwarz, W., Trans.). Milano: Editrice Antroposofica.

Steltzner, M., & Williams, M. (1988). The evolution of interface requirements for expert systems. In J. A. Hendler (Eds.), Expert systems: The user interface (pp. 285-306). Norwood, N.J: Ablex.

Stenlund, S. (1988). Undersökningar i matematikens filosofi. Stockholm: Thales.

Stevens, G. C. (1982). O.R.workers, information systems analysts and the challenge of the micro. J. of the Operational Research Soc., 33, 921-929.

Strachey, C. (1966). Systems analysis and programming. Scientific American, 215(3, September), 112-127.

Swanson, E. B. (1989). Information system implementation and organizational health. In C. W. Churchman (Eds.), The well-being of organizations (pp. 179-187). Salinas, Calif.: Intersystems Publications.

Sørgaard, P. (1986). Evaluating expert system prototypes. In H. E. Nissen & G. Sandström (Eds.), Quality of work versus quality of information systems (Report of the Ninth Scandinavian Research Seminar on Systemeering, Båstad, 19-22 August 1986) (pp. 187-201). Lund: Lund University, Dept. of Information Processing.

Sørgaard, P. (1988). A discussion of computer supported cooperative work No. Doctoral dissertation). Aarhus University, Dept of Computer Science.

Utter, D. F. (1982). Writing requirements for interfaces between computers. IEEE Trans. on Communications, COM-30(6, June), 1276-1280.

Utter, D. F. (1984). Properties of the system design through documentation (SDTD) methodology. In First Int. Conference on Computers and Applications, Peking, China, June 20-22 1984 (pp. 809-814).

van Gigch, J. P. (1988). Design of the modern inquiring systems - II: The contemporary computer. Systems Research, 5(3), 269-271.

van Stigt, W. P. (1979). The rejected parts of Brouwer's dissertation on the foundations of mathematics. Historia Mathematica, 6, 385-404.

Verene, D. P. (1982). Technology and the ship of fools. Research in Philosophy and Technology, 5, 281-298.

Wasserman, A. I., & Gutz, S. (1982). The future of programming. Comm. of the ACM, 25(3, March), 196-206.

Webb, J. C. (1980). Mechanism, mentalism, and metamathematics. Dordrecht: Reidel.

Wegner, P. (1982). Reflections on capital intensive software technology. ACM-SIGSOFT Software Engineering Notes, 7(4, Oct.), 24-33.

Weil, S. (1970-1974). Cahiers [Notebooks] (3 vols.). Paris: Plon.

Weizenbaum, J. (1976). Computer power and human reason. San Francisco: Freeman.

Weyl, H. (1949). Philosophy of mathematics and natural science. Princeton: Princeton University Press.

Whitaker, R., & Östberg, O. (1988). Channeling knowledge: Expert systems as communications media. AI & Society, 2(3), 197-208.

Winograd, T. (1979). Beyond programming languages. Comm. of the ACM, 22(7, July), 391-401.

Winograd, T. A., & Flores, F. (1986). Understanding computers and cognition: A new foundation for design. Norwood, N.J: Ablex.

Wirth, N. (1976). Algorithms + Data structures = Programs. Englewood Cliffs: Prentice-Hall.

Wood, B., & Doyle, K. (1989). Doing the right thing right: An exploration between soft systems methodology and Jackson's system development. In Proc. of the ISSS Int. Society for the Systems Sciences, 33rd Annual Conference, Edinburgh, Scotland, 2-7 July 1989. Vol 1 (pp. 146-155).

Yeh, R. T., (Ed.) (1977). Current trends in programming methodology. Englewood Cliffs: Prentice-Hall.

Zellini, P. (1988). Humanistic and ethical aspects of mathematics No. Report UMADP-RRIPCS-4.88). Umeå University, Inst. of Information Processing.

Östberg, O., Whitaker, R., & Amick III, B. (1988). Den automatiserade experten. En uppsats om expertsystem och några intryck från en AI-konferens. No. Stockholm-Farsta, Sweden: Swedish Telecommunications Administration.