To appear in BIT Vol 28 (3), 1988

November 1987

Version August 13, 1988

DATALOGY - THE COPENHAGEN TRADITION OF COMPUTER SCIENCE

Edda Sveinsdottir
Datalogisk Afdeling
Roskilde Universitetscenter
Postbox 260
DK-4000 Roskilde, Denmark

and

Erik Frøkjær
Datalogisk Institut
Københavns Universitet
Universitetsparken 1
DK-2100 København Ø, Denmark

Dedicated to Peter Naur on the occasion of his 60th birthday

Abstract.

Since the middle of the 1960s, computer science has been practised in Denmark under Peter Naur's term *datalogy, the science of data and data processes.* Starting at Regnecentralen and the University of Copenhagen, the Copenhagen Tradition of computer science has developed its own special characteristics by means of a close connection with applications and other fields of knowledge. The tradition is not least visible in the area of education. Comprehensive project activity is an integral part of the curriculum, thus presenting theory as an aspect of realistic solutions known primarily through actual experience. Peter Naur early recognized the particular educational challenges presented by computer science. His innovations have shown their quality and vitality also at other universities. There is a close connection between computer science training as it has been formed at Copenhagen University, and the view of computer science which has characterized Peter Naur's research. We illustrate how the study of programming and system development conceived as a human activity has been an all-pervasive theme in Naur's work. This approach has set the scene for central research issues in software development which today seem more topical than ever.

CR categories: K3, A0, D0.

Keywords and phrases: computer science education and training, computer science, datalogy, programming methodology, software engineering, Peter Naur, history.

1. Introduction.

Computer science is still a new discipline, and comparisons with different scientific traditions are both important and valuable. We will here report on the development of computer science at the University of Copenhagen, and in particular illuminate Peter Naur's part in this development. Using Naur's term "datalogy" - which we in Denmark have adopted as the name

for computer science - the discipline has developed its own special form at Copenhagen University, here named the Copenhagen Tradition. This tradition is present at other Danish universities as well and is especially visible via its practice within education.

Peter Naur has played a central role in developing the Copenhagen Tradition. In addition to his pioneering work in programming language design and software development methodology, throughout the years Naur has been extremely concerned about reaching an understanding of computer science and from that deducing the implications for content and form of computer science education and training. One central insight is that the essence of the discipline can be learned only through experiencing the processes associated with it. This is primarily achieved through students participating in comprehensive projects. These ideas have been developed further at the university in collaboration with colleagues and students, and have contributed significantly to the creation of an original program of study in computer science, which has proved its value through its applicability to systems design and construction, a field which is still characterized by a scarcity of skilled people and many unsolved problems.

From the beginning the emphasis has been on eliciting the underlying ideas and principles of computer science understood as "the science of the nature and use of data" called datalogy [41]. The central theme is programming including all the activities involved, i.e. analysis of requirements, design, implementation and maintenance of program systems with an overall awareness of the interplay between the people involved, the computers, and the organization. Thus, the word programming in this paper should be understood in a much broader sense than just coding and testing for a specific design, as it is often used in the context of commercial programming.

We shall further illustrate the Copenhagen Tradition by discussing some selected topics within the field of software development:

- the computer as a tool for people in problem solving
- structured programming and formalization in software development
- large data systems and the proper place of software development methods
- understanding programming as theory building.

Peter Naur has made contributions of fundamental importance to computer science in all of these areas.

2. How the Copenhagen Tradition was formed.

During the sixties computer science at Copenhagen University developed through a process of close cooperation between the outstanding research and development environment at Regnecentralen, originally an institution under the Danish Academy of Technical Sciences, and individuals from the Institute of Mathematics at the university.

At Regnecentralen a group of people, predominantly mathematicians and electrical engineers, had been gathered together under the leadership of Niels Ivar Bech. The group was engaged in research and development that led the international field, both with respect to

programming languages through the Algol 60 effort [35, 67] and with respect to the construction of electronic computers and systems programs [23, 36]. A number of young scientists and programmers were trained in this programming environment at Regnecentralen. Per Brinch Hansen [4, 5] and Charles Simonyi [33] are well-known outside of Denmark. The group at Regnecentralen was aware of the general utility of computers, and it was realized that there would be a great demand for qualified people trained in the emerging area [34]. It was decided to have key persons try to extract a kernel of knowledge from the variety of assignments and problems they worked on. Already in February 1962, a one-year curriculum in administrative data processing was started at Regnecentralen with the dual purpose of training employees at the institution and developing educational material for general use. The program was developed in a collaboration with two other Scandinavian computing centers, Norsk Regnecentral (Oslo) and Matematikmaskinnämnden (Stockholm). Students were required to have a graduate university education in engineering, economics, or other disciplines. This educational program was very early as can be seen from a panel discussion at the IFIP 1962 meeting, where also a graduate program in computer and information sciences at the Moore School of Electrical Engineering, University of Pennsylvania, is mentioned [18].

Visions at Regnecentralen of bringing the use of computers into the universities on a large scale took form through cooperation with the Technical University of Denmark. The perspective of these plans, however, was not understood, and their realization was blocked both by forces within the state administration and by the acceptance of a large computer presented as a gift from IBM to the Technical University. The story of the early development at Regnecentralen is reported in [82]. It should be mentioned that a number of key persons in the development at Regnecentralen later joined the staff of various universities. In addition to Peter Naur, these include Christian Gram and Henning Isaksson at the Technical University, H.B. Hansen at Roskilde University Center, Christian Andersen at Aarhus University, Søren Lauesen at the Copenhagen Business School, and Per Brinch Hansen presently at Syracuse University, New York.

The history of computer science at the university.

At the university there was a growing realization that the fundamental nature of computer science made it important to establish a framework within which the new discipline could develop. In 1963 the new computer, Gier, from Regnecentralen was installed at the Institute of Mathematics in the Section of Information Processing. This created the basis for the development of computer science at Copenhagen University. One of the people behind the construction of Gier, the mathematician Bjarner Svejgaard of the Danish Geodetic Institute, was appointed head of the section with the responsibility of taking care of teaching and research in connection with the new computer. Researchers from the fields of chemistry, physics, psychology, linguistics, botany, economy and others collaborated with the section. Working together with Bjarner Svejgaard, individuals from these various fields acquired an inspired and deep understanding of the new tool.

Today we can see that many of the seeds sown at that time have grown into creative research environments.

Students of mathematics with special interest in computer science participated in these interdisciplinary projects. In the Mathematics 4 course the Gier computer replaced the electromechanical calculators previously used for numerical computations. The course soon included an introduction to programming using the new programming language Algol. In addition, more advanced subjects were taught in connection with the ongoing computer science research. In 1968 it was decided to begin a master's program (cand. scient.) combining computer science and mathematics with the possibility of substituting subjects from other fields for mathematics. The one-year courses, Datalogy 1 and Datalogy 2, started in the summer of 1969 and 1970 respectively as a continuation of Mathematics 4, which later had its name changed to Datalogy 0 and became the basic introductory course in datalogy.

Outside the Institute of Mathematics the EDP-committee of Copenhagen University sat its mark on the early development and in 1968 brought forth plans for the future computing center at the university as well as a proposal for a future Institute of Datalogy. At this time the environment created by Svejgaard had suffered a set-back when he accepted the position of director of Nørrejysk Datanet, later to become RECAU, the regional EDP-center at Aarhus University [6]. His leaving was a reaction to the fact that the university had so far not established the necessary environment in which computer science could grow. This meant a crisis for computer science at the university as expressed in an editorial in the university paper by the three remaining academic staff members, the late Peter Lindblad Andersen, a brilliant and original scientist who unfortunately died prematurely in 1973, Peter Møller-Nielsen, and Edda Sveinsdottir [2].

Realizing that the situation was critical, Copenhagen University seized the opportunity of calling Peter Naur to Denmark's first chair in computer science. After thorough deliberation and negotiations with the university to obtain realistic conditions for the new institute [37], Peter Naur accepted the offer in July 1969. Thus the basis for DIKU (Datalogisk Institut ved Københavns Universitet) was established, and the institute could start its work in April 1970. Figure 1 shows how the new professor invited teachers, administrative staff, and a number of students to take part in an open discussion of the theme "what do we think is essential for the institute we are about to create?". The new institute developed in an inspiring environment of collaboration involving everybody, see Figure 2.

The three datalogy courses mentioned above constituted and still constitute the first part (bachelor-level) of the master's program in computer science. The second part consists of two years of further study in computer science. The form of teaching is different from what is traditionally found at the university. The differences include heavy emphasis on project work, where students, in groups and individually, solve rather large problems under the supervision of teachers and advanced students. Through their project work, which occupies half of their study time, the students get the opportunity to use the theoretically learned material to solve actual problems. At the same time professional skills such as cooperation, project planning, project

implementation, testing, documentation, and report writing are learned. Regarding the project reports, the importance of careful argumentation and descriptions in prose of the development process and its results are emphasized.

At Copenhagen University computer science is placed in the natural science faculty. However, it has always been characteristic of the pattern of study that computer science is combined with many different subjects outside the faculty and even outside the university. This is described by the fact that 1/3 of the students combine computer science with subjects from the natural sciences (mainly mathematics), 1/3 with subjects from the humanities and the social sciences, and 1/3 with subjects from the Copenhagen Business School. Nearly 20 percent of the students are women. As of today 233 students have graduated from DIKU with a degree of cand. scient. (master of science) in datalogy. About 1300 students are currently studying for a major degree in computer science and about 800 for a minor [14]. Figures for the number of students having left the institute with a minor are not available. Some statistics for employment of computer science candidates in Denmark can be found in [31].

(Insert Picture)

Figure 1. Peter Naur invites a group of colleagues and students to his home at Begoniavej to discuss the creation of the coming Institute of Datalogy. [*Reconstructed from the Danish original.*]

An early report [25] contains interviews with the teachers at DIKU giving their views on problems in education and training in computer science. In a paper by Peter Johansen entitled "*Datalogy*", published on the occasion of the 500th anniversary of the founding of Copenhagen University, a broad description of the Institute of Datalogy and its research is presented [28]. DIKU was originally given an independent administrative position outside the traditional faculties at the university. In 1976, however, the institute was placed administratively under the natural science faculty against the will of its teachers and students, as they felt it important to preserve an independent position with freedom to collaborate on an equal footing with other disciplines at the university. A discussion of the placing of computer science disciplines at Copenhagen University is given in [81]. The institute has developed under difficult conditions with respect to resources, and from the beginning it had many students and too few academic staff members. At times the situation has been very critical and has necessitated extraordinary efforts and funding [79].

(Insert Picture)

Figure 2. The birth of the institute in the wake of the student uprising of 1968 was marked by a unique collaboration between students, teachers and staff. The picture of the studies committee from 1970 include from top left, Peter Johansen, Peter Jensen, Ole Caprani, Bo Vincents, Peter Naur, Edda Sveinsdottir, Nils Andersen, Lissi Olsen, Steen Jürs, and Jesper Gørtz. We have included Flemming Sejergaard Olsen as an insert to the left in the picture in memory of his fine contribution as director of DIKU in 1972-1973, the first and only student in such a function in the 500 year history of Copenhagen University.

3. Peter Naur's view of computer science in education.

From the middle of the sixties, Naur developed his ideas of the new subject called datalogy and its place in education. Naur's own background in computing goes back to 1951 when as a young astronomer, 22 years of age, he visited Cambridge, England, and worked with the EDSAC, the first practical, stored program computer to be put into operation [65]. Later he was on the staff of Regnecentralen from 1959-1969. His thoughts on datalogy can be found in a booklet published in Danish in March 1966, "*Plan for a course in datalogy and datamatics*" [43]. In this work Naur systematically discusses the fundamental concepts of datalogy which comprise data, data representations and data processes, and he makes clear the generality of these concepts. Under the term datamatics (that part of datalogy which deals with the processing of data by automatic means) more practical and technological aspects are treated.

At this time the contents of a new university subject called "computer science" were being discussed in university circles in the United States. The ideas for an undergraduate program in computer science were described in the form of preliminary recommendations from the ACM Curriculum Committee on Computer Science [83], and had already been put into practice at some American universities. In Sweden plans for a university degree program in administrative data processing had been proposed [15]. In contrast to these early plans made in other countries, Naur's plans had to do not only with the university education of computer specialists, but also the school education of young people from an early age. Until then the interdisciplinary aspect of the teaching of computing had primarily consisted in the use of specific programming languages. In contrast Naur saw a great need for the elementary teaching of datalogy to be more comprehensive in content.

Computing studies in the general educational system.

Naur saw that the basic concepts of computer science are of a general nature and in many ways throw new light on everyday tasks and on different areas. From this follows a great need for education, and Peter Naur felt it necessary to share this insight with the general public. He wrote about these matters in teachers' journals, gave examples of teaching material, and talked on the radio about the new field [44, 46, 49, 52]. In 1966 Naur received the Rosenkjær-prize for his efforts. In his Rosenkjær lectures, which were given on the radio and also published as a book [45], Naur discussed the wide-reaching consequences of computer science:

> "Thus if one has realized that on the one hand computer science unites a large number of vital human activities and concepts in one overall point of view, and on the other hand is capable of inspiring and renewing ideas in an equally great number of subjects, one can be in no doubt as to its place in general education. To arrive at a reasonable view of this position, one must of course make comparisons with similar disciplines and one will arrive at the conclusion that language and mathematics lie closest. Computer science and these two subjects deal with signs and symbols as tools invented by human beings. The three subjects have also in common the fact that they are tools for many other subjects. Therefore they enter education in two ways; both as auxiliaries in the study of many other subjects and as majors where specialists are trained in these subjects

themselves. All of us have had to learn a considerable amount of languages, arithmetic and mathematics in school, without many of us subsequently becoming linguists or mathematicians. In the same way we must bring computer science into the school and prepare ourselves for life in the era of the computers, just as reading and writing are regarded as necessary prerequisites for life in a society characterized by the printed word.

Computer science can enter the school curriculum either as an independent subject or as a part of mathematics; what is crucial is the contents of what is taught. The emphasis should be on data, data representations, and data processes. These fundamental concepts must be illustrated by means of a number of concrete examples which can easily be taken from areas that the pupils are familiar with - arithmetic, spelling, looking things up etc. Simple examples of the use of mechanical and electrical phenomena for data representation should also be included. These examples should be illustrated by simple experiments. Computers should also be mentioned, not as the most important element in the subject, but rather as a final point of information." [*Translated from Danish.*]

The general qualities in working with computer science in education are illustrated in the following quotation, where the understanding of programming is accentuated in relation to fundamental human activities such as learning and problem solving [45]:

"Computers make inexorable demands as to a complete description of the processes they are to carry out. At the beginning this is a new and difficult burden for those who have to solve the problems. However, in many cases this new demand for precision has proved to be a definite plus by virtue of the clarity of thought it brings in its wake. Through working with absolutely formalized descriptions of our subject matter, we become familiar with it in a new and inspiring way, we become aware of possibilities we never experienced before, and we realize how superfluities can be removed with a corresponding gain in clarity and economy.

(...) In the first place, it should now be clear that programming a computer is mainly a question of planning. A program is simply a detailed plan of what a computer has to do, how it is to behave in relation to all the eventualities that may occur while the data processes are being carried out. The practical job of programming a computer is the best possible training in how to plan. Human limitations with regard to this type of work are something that has been experienced and which should be stressed. It is quite in the order of the day that a program delivered by a programmer, even a master in the field, contains faults. These may be trivial, but of such a nature that when the program is being tried out in the computer, it runs wild. Let this fact serve as a memento to all those who would have us believe that all problems can be solved by more planning." [*Translated from Danish.*]

In spite of Naur's strong feeling about the need for teaching computing as a part of general education, he doubted that this goal could be reached easily or quickly. Thus, in 1966 Naur writes that he foresees that it will take decades for the necessary changes to take place because of the professional and organizational inertia of our educational system [44]. So far two decades have passed and the introduction of computing studies with a relevant content into the general educational system is in our view years away.

Computer science at university level.

Seen in an international perspective, Naur's view of datalogy as a separate and independent

science was original and among the earliest. Here it is relevant to mention S. Gorn [22] and especially G. Forsythe [21] who was much in line with Naur when emphasizing the importance of extracting and teaching the essence of computer science. Naur presented a survey of the central topics in a systematic course in computer science at the IFIP congress in 1968 [47], see Figure 3.

Section 1.	*Basic concepts and methods*
	1.1. Data and data processes
	1.2. Computers and programming languages
	1.3. Proof, test, and construction of data processes
Section 2.	*Processes on single data items*
	2.1. Digital data representations
	2.2. Choice processes
	2.3. Numbers and arithmetic
Section 3.	*Processes on intermediate quantities of data*
	3.1. Searching and sorting
	3.2. Sequential analysis of linear texts
	3.3. Evaluative notations
	3.4. Lists and links
Section 4.	*Man/machine communication*
	4.1. Input to computers
	4.2. Output from computers
	4.3. Conversational techniques
Section 5.	*Processes on large quantities of data*
	5.1. Backing stores
	5.2. Administration of backing stores
	5.3. Sorting on magnetic tape
Section 6.	*Large data systems*
	6.1. Elements of large data systems
	6.2. Design of large data systems

Figure 3. A survey of the essential topics in a systematic course in computer science according to Peter Naur. The ideas were presented in Europe at the IFIP Congress in 1968 and, by invitation, a month later at the ACM '68 National Conference in the United States.

The ideas were developed in the years 1966-1970 by a group at Regnecentralen working on comprehensive course material covering the range of topics found in the "*Plan for a course of datalogy and datamatics*" [43]. The group consisted of Christian Gram, Jens Hald, H. B. Hansen, Peter Naur and Alan Wessel. When the group split up in 1970, a series of 11 booklets covering a wide range of topics in the course outline had been published.

At the IFIP congress Naur discussed his ideas about an introductory university course in datalogy in relation to "*Curriculum 68: Recommendations for academic programs in computer science*" [7], a proposal that was to have great influence on the development of computer science especially in Anglo-Saxon countries. Naur stressed some important differences:

> "The deepest difference is that the ACM Curriculum seems to make an attempt to cover the field in an almost encyclopedic manner, apparently making sure to mention all the current techniques, languages and practices, however briefly, while the datalogy course strives to emphasize the underlying ideas and principles, while omitting many particular instances of the various notions. (...) By keeping these matters out of the

course of datalogy, this can concentrate on basic matters, common to all environments."

In spite of the encyclopedic character of Curriculum 68, Naur was able to point to basic topics not treated, for example the need to consider human reactions, man/machine interaction, and a topic such as choice processes to represent arbitrary transformations.

Another difference was the way in which the different topics were treated. The construction of compilers was an example. Where the ACM Curriculum stressed the processing of programming language texts, Naur saw compilers primarily as examples of large data systems [47]:

> "In constructing a compiler we need a multitude of techniques, including most of those discussed in the earlier sections of the course. It seems highly desirable that these techniques be presented as methods, not only of compiler writing, but of general utility. Compilers as such should be discussed as examples of large data systems, at a stage when the individual problems of detail have already been treated. More details of this approach may be found in [*our ref. 42*]."

In [59] Naur summarized his critique of the ACM Curriculum 68:

> "Generally speaking, the ACM Curriculum 68 tends to stress those parts of the subject that lend themselves to formalization. It is weak in such aspects as are related to the experience and intuition of the practitioner, such as man/machine interface problems, the psychology of programming and of project work. Also, applications are practically absent from the curriculum."

In 1974 Naur published his "*Concise survey of computer methods*" [57] which is a comprehensive treatment of datalogy. This book is a valuable source of fundamental ideas and methods in computer science and contains timeless guidelines on how to proceed in program design, coding and testing. Along with the technical material it includes a treatment of social aspects as well. The book inspires the reader to further systematic study of computing literature also via its many references to Computing Reviews.

Problems in computer science education and training.

Naur has pointed out the importance both of proficiency in practical work with computers and data media, and in the methods of project work including documentation techniques and maintaining standards. Such matters cannot be dismissed as trivial when we want to stimulate the students to carry out their own investigations and to gain their own experiences within the field. Accordingly, their training must emphasize a concrete understanding of forms of data and data processes on the one hand and a work methodology on the other hand. This work methodology includes both problem solving techniques and the systematics of project development. Particularly important issues are problem formulation [54] and project organization including aspects of human relations within the project group. Project oriented work techniques must be learned early in the course of study. Only if the students have learned suitable techniques for solving smaller problems is there hope that they will be able to cope professionally with large projects. Peter Naur has described the issues in his Leonardo Fibonacci Lecture, "*Project activity in computer science education*", delivered in Pisa in April 1970 [48].

Naur's ideas about the teaching of datalogy and the development of constructive abilities in computer work display a fundamental tension in our field which constantly crops up. Peter Denning, editor-in-chief of Communications of the ACM, addresses the issues in his editorial on the occasion of the 40th anniversary of the ACM in 1987 in a way which is very similar to Peter Naur [9]. Denning relates how different attempts over the years at basing computer science on the traditions of either science or engineering, never proved fully satisfactory. Through a discussion of the experimental process as it is known from science, and the design process as it is known from engineering, Denning concludes that in computer science the two processes are intricately intertwined.

This intertwining of experimentation and design was recognized by Naur at an early stage as stated for example in his study of a program development process [55] and in "*Design and development of large data systems*", chapter 18 in [57], where he treats the many problems entailed in the development of large data systems.

Computer science in relation to other disciplines.

Computer science should be utilized by - and receive inspiration from - other fields. Fundamentally, data and data processing can be regarded as tools to be used by people in their many and varied activities. As with mathematics and linguistics, the field of computer science was developed around a tool of a great generality created by man, namely data. It was with these notions in mind that, from the beginning, Peter Naur took it for granted that computer science would be introduced into the curriculum of all students at the University of Copenhagen within a foreseeable future [53].

Peter Naur pointed to the possibility that the teaching of computer science in other disciplines like the humanities, the social sciences, law school, medical school and other fields in the natural sciences, could be carried out by people from these fields with the support of the computer scientists. In this way contact between computer science and the other disciplines at the university would increase, and the tendency for computer science to develop in a self-sufficient manner could be avoided (see also [60]). It was proposed that computer scientists and teachers from the other fields of study should create course material so that the computer scientists provided the terminology and the systematic treatment of the relevant topics of datalogy, while illustrations, examples and exercises were provided by the teachers of the respective fields of study. In 1973 some academics at the university came with concrete proposals for course material suitable for teaching computer science as an auxiliary subject within other fields [1]. The aim of the material was in accordance with the proposals of the Johnsen Committee which presented the first comprehensive plans for introducing computing into education in general in Denmark [29]. Both recommendations include a limited but important element of programming - and thus on this point represent an alternative to the current approach to be found in the school system in Denmark, where the importance of concrete experience of programming, as distinguished from coding, is still not commonly understood.

4. The Copenhagen Tradition of software development methodology.

Research in computer science in the Copenhagen Tradition can be illustrated in many ways. We have chosen to draw attention to central themes within software development methodology, where Peter Naur has been a central figure in international scientific discussion.

The computer as a tool for people in problem solving.

An early stage is the work on the development of software tools. Peter Naur's role in the definition of Algol 60 is well-known. Algol 60 represents a milestone in the development of programming languages, both in terms of the concepts introduced and the first recognized use of formal notation for syntax, now known as the Backus-Naur form (BNF) introduced by J. W. Backus and given a final form in the "*Report on the Algorithmic Language Algol 60*" [35]. Most theoretical and much practical language and compiler work since 1960 has been based on Algol 60. A fascinating account of how Algol was developed can be found in "*History of programming languages*" based on a conference held in 1978 [67].

At Regnecentralen the design of Algol 60 was immediately followed by the development of compilers for Algol 60, in several versions, for several computers, during the years 1960 to 1967, and for Cobol during l963-l965. Also this work by Naur and the group at Regnecentralen is of a pioneering character. The Gier-Algol compiler - called a masterpiece by E. W. Dijkstra - was based on viable new ideas, such as automatic storage allocation during program execution (later known as paging), multipass translation, and pseudoevaluation of expressions [36].

With these experiences as a background Peter Naur discusses software tools from a perspective in which people, tools, and problems are seen as a unified whole (see Figure 4).

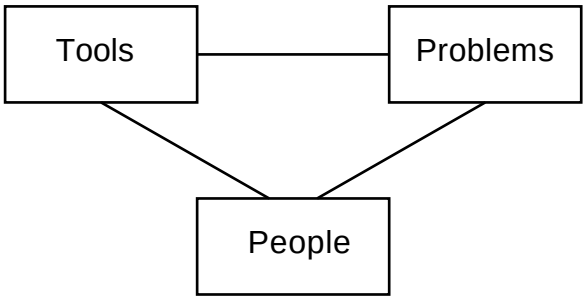

Figure 4. The basic components of problem solving situations according to Naur [39].

Illustrating by carefully selected examples Naur discusses how people's understanding and formulation of problems are closely connected with the tools at their disposal [39]. The situation in Figure 4 is completely symmetric:

"We can view it from the position of either of the three elements and discover that the two remaining have a subtle relationship: People can only understand the problems on the background of an assumed set of

tools, while a tool designed to solve a problem which is understood by nobody is meaningless. Problems exist only in the minds of people and only relative to understood tools. Tools only exist as such in so far as some people think of them as the proper things with which to solve some problems."

Given these considerations, a general platform is formulated from which Naur can continue the discussion of the characteristics of programming languages as tools for people when solving problems from different areas.

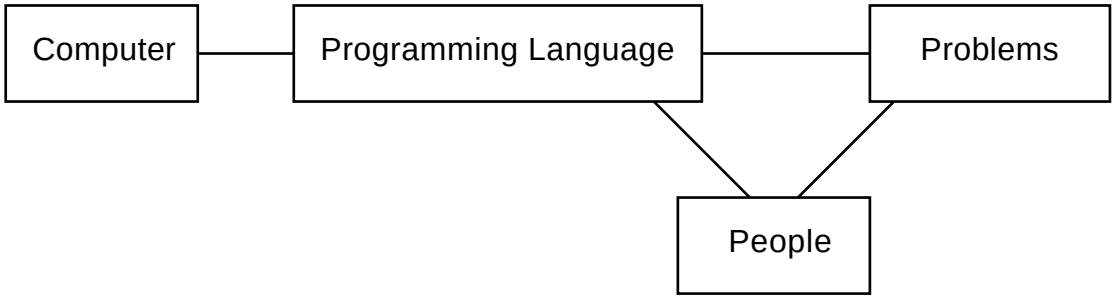

Figure 5. The fundamental components in problem solving using computers according to Naur [39].

Many fundamental research themes can emerge from a discussion of the interplay between the components of figure 5. In [39] the following are pointed out:

"For good or bad, the characteristics of the programming language will shape the thinking of people and their conception of problems. The dangers of this are at once apparent when we realize that programming languages are designed to be, as it is said, problem-oriented. Now, in the view I put forward here there is no problem without an understanding of the tool available for solving it. Being problem-oriented must therefore imply an understanding of a tool also. What crops up here is of course the conventional view of the problem, the view based on older tools. The great danger of problem-oriented languages is therefore that they will tend to perpetuate a view of the problem which is appropriate to an obsolete tool."

Illustrations of the utility of this perspective can be found in [56, 60, 63]. As a general theme it is emphasized that empirical studies are of fundamental importance in shedding light on the qualities and difficulties in the interplay of datalogical tools, problems, and people. This is a theme that Naur returns to again and again (see for example [69, 73, 76]).

From this view of software development and Naur's later work where he reaches an understanding of programming as theory building [72] (treated in a later section), it follows that the design of computer support for people in work processes cannot be merely reduced to the design of data processes. Important issues cannot be formally described, but must be investigated through experimental design and use. This viewpoint is very much in line with an emerging understanding of system development found in works by, for example, H. Dreyfus and S. Dreyfus [16], P. Ehn and M. Kyng [17], C. Floyd [20], and T. Winograd and F. Flores [84].

Structured programming and formalization in software development.

In the discussions about design techniques in the late sixties it was Naur's empirically founded insights and investigations which formed the basis of a critique of and corrective to Dijkstra's ideas on structured programming [13], and Wirth's ideas on stepwise refinement [85], as techniques for developing safer and more modifiable large data systems. The central paper here is "*An experiment on program development*" [55], where Naur put forward the view that the strict top-down approach advocated by Dijkstra and Wirth could not be expected to work as a general method. To Naur their ideas did not seem right as guidance for the programmer during problem solving. Program development styles should allow for personality factors and Naur asked for more observations. The paper gave a careful illustration of a method of conducting such observations of program development based on diaries. Methodological discussions and results from this line of empirical research can be found in [58, 66, 70, 76]. In the following years the discussion about structured programming continued. A survey of this discussion and the different interpretations of the concept of structured programming can be found in Infotech's State of the Art Report from 1976 (see [61]), where we find that Peter J. Denning in particular was able to give the subject a more complete treatment [10].

Though Naur was critical of the ideas of structured programming presented by Dijkstra and Wirth, he was strongly concerned about the objective to obtain more reliable and maintainable programs, especially in connection with large data systems.

In the paper "*Proof of algorithms by general snapshots*" published in BIT in 1966 [40], Naur used program invariants, which he called "snapshots", and advocated correctness proofs of programs and described a technique for specifying them. Naur also emphasized that the use of invariants constituted a useful technique whereby programs could be constructed. This was actually the first article on proving programs correct, as mentioned by D. Gries in his book, "*The science of programming*" [24]. In close relation to the proof problem, the problem of program construction gave rise to the suggestion by Naur, that in order to establish the connection between the requirements to be satisfied by the program and the program itself, it should be possible to join logically related statements together to form action clusters [51]. For this to be done effectively, the programming language should include special clauses for delimiting the clusters. The concept of clusters is quite similar to the ideas on modular programming developed by Parnas - see the historical note in [78] - though Naur does not explicitly emphasize the important principle of information hiding of the module concept.

Structured programming and design techniques were brought to practitioners by method experts like M. Jackson, E. Yourdon and T. DeMarco [26, 27, 86, 87, 8]. To many programmers the ideas of structured programming were reduced to a question of avoiding **go to** statements in the programs. A background for this is the cogent paper in 1968 by Dijkstra with the alarming title "*Go to statements considered harmful*" [11]. Ironical as it may seem, it was actually Peter Naur who first pointed out a number of unfavorable effects of the unrestricted use of **go to** statements. Donald Knuth tells the story in his paper "*Structured programming with go*

to statements" [32]. In 1963 Naur wrote the following in BIT's section on Algol programming with the title "*Go to statements and good Algol style*", drawing attention to the teaching of programming as a source of the problems [38]:

> "Now why is this so ugly? Because it is inelegant and uneconomical. A **go to** statement is about the most empty construction conceivable. Executing it is like being told: It is not me, it is my colleague! I myself find this most frustrating when I read ALGOL programs. (...) Well, if you look carefully you will find that surprisingly often a **go to** statement which looks back really is a concealed **for** statement. And you will be pleased to find how the clarity of the algorithm improves when you insert the **for** clause where it belongs.
>
> There seems to be little doubt that this misuse of the **go to** statement must reflect the teaching of programming and it seems likely that it is related to the great stress that is placed on the use of flow diagrams in some circles. Somehow the feeling seems to be prevalent that the flow diagram is in some sense more basic than other programming notations. I would like to take the opportunity of opposing this view. While it is true that for some years the flow diagram had a unique position by being in fact the only common, machine-independent programming language available, these times are now past, fortunately I would say. If the purpose is to teach ALGOL programming the use of flow diagrams will do more harm than good, in my opinion."

Although Naur very early recognized the importance of programs having a simple structure, he correctly foresaw that the methods developed later that came to be known as structured programming were not in themselves a sufficient answer to the problem of producing correct programs in the real world.

In recent years other suggestions for structured methods have been the so called formal specification methods, of which a part of the scientific community has great expectations seeing them as an answer to the continued problems in software development. Again Naur has delivered a penetrating critique emphasizing that only formal specifications which contribute by supporting the intuitive understanding of the matter at hand can be recommended, see the paper "*Formalization in Program Development*" [69]. At the TAPSOFT conference in Berlin 1985 around the theme "*Formal methods and software development*", Naur's contribution, "*Intuition in software development*", stood out and stimulated essential discussions on account of his severe criticism of the fundamental assumptions that underlie various formal methods [73]:

> "The major conclusion of the present discussion is that software development in all its phases, and irrespective of the techniques employed in its pursuit, must and will always depend on intuition.
>
> The fundamental way of reducing failures of human intuition is to apply multiple work and check. Rules for guiding the software development depend on intuition to decide where and how they apply. Consequently a view of software development that makes the application of rule-based methods and notations the basic issue is misguided. The deeper problem of software development is the programmer's building of theories of the computer-based solutions."

In the paper Naur attacks the belief in rule governed behaviour of the human mind, that lies behind the claim that there is any proper or correct programming methodology. Much work in

artificial intelligence research or cognitive science is based on a similar notion, i.e. that the human mind works in a rule governed manner like an information processor. This leads to naive expectations of the possibilities for controlling or replacing human activity by computers [74, 75].

Large data systems and software development methods.

Another reason for the scepticism of researchers at Copenhagen University about structured programming and other general methods of software development, was their personal experiences with the development of large software systems outside the universities. A decisive event for understanding the special problems encountered in large software systems took place at the Nato Conference on Software Engineering in Garmisch in October 1968. This conference created a sensation because for the first time there was an open admission of a software crisis, as mentioned by Dijkstra [12]. The report from the conference provides a first source of insight into the problems of software engineering, a term coined at this conference. Also it is here that the discussions on structured programming seem to have begun [50].

Naur has shown great interest in software engineering and programming methodology and has treated many different aspects of the problems of the design and development of large software systems. This can be seen from the titles of papers like "*Software reliability*", "*Prospects for the programming methodologies*", "*Control record driven processing*", "*Diminishing returns of user programming*", "*Programming studied from case activity records*", and "*Programming as theory building*". His views have been consistently based on empirical studies and have been presented in forums of practitioners and researchers, as can be seen from the references [62, 68, 61, 64, 76, 72]. A survey of the problems and methods of solution in the field of large data systems is given in chapter 18 in [57]. As an alternative to the usual recommendations of a software life cycle type of development, Naur points to an approach where the development problem is regarded as primarily one of overall design reached through an experimental attitude. In many ways his discussion anticipated the useful techniques we later in the eighties learned under the heading of prototyping in software development. Many of these techniques can be picked up in "*Concise Survey of Computer Methods*" [57] and Naur's awareness of this experimental approach can be seen in a comment on a paper by A. Helms Jørgensen [30].

Understanding programming as theory building.

Still today our profession has serious problems in meeting the demands for more and better software systems and according to Naur we must not expect these difficulties to be overcome mainly through new methods in software development. This is one of the conclusions of the paper "*Programming as Theory Building*", where a thorough treatment of programming viewed as a human activity is given [72]. Instead the key issue in programming is

> "… to have the programmers build a theory of the way the matters at hand may be supported by the

execution of a program. Such a view leads to a notion of program life that depends on the continued support of the program by programmers having its theory. Further, on this view the notion of a programming method, understood as a set of rules of procedure to be followed by the programmer, is based on invalid assumptions and so has to be rejected. As further consequences of the view, programmers have to be accorded the status of responsible, permanent developers and managers of the activity of which the computer is a part, and their education has to emphasize the exercise of theory building, side by side with the acquisition of knowledge of data processing and notations." [72]

Naur's concept of theory is inspired by Gilbert Ryle, the central figure among the Oxford philosophers in the middle of this century [80]. A theory in their sense is not a formula or a specification that can be put down on paper, but resides in the head of the programmer who is building a theory about the part of reality relevant to the program as he or she goes through the software development process. Any program will only reflect a limited part of such a theory. In short:

"a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern." [72]

The paper gives in-depth explanations of why so many attempts with different system development methods in the last decades have failed, powerless when faced with the practical problems of developing large and complex data systems. The central idea is that when the system developers have a clear idea of the software system in its fullest detail, then many systematic methods and forms of documentation can be used, provided they are mastered by the system developers. Without this detailed idea or understanding, the methods are of no help. The criteria for the selection and combination of techniques are that they should support the intuitive understanding of the programs involved and the part of reality they model.

Naur's theories about programming have important consequences for system developers and managers. The extensive division of labor we have known for many years in our field, especially between system analysts and programmers, impedes the development of good systems, and a reorientation of the programmers' training is needed:

"The raising of the status of programmers suggested by the Theory Building View will have to be supported by a corresponding reorientation of the programmer education. While skills such as the mastery of notations, data representations, and data processes, remain important, the primary emphasis would have to turn in the direction of furthering the understanding and talent for theory formation. To what extent this can be taught at all must remain an open question. The most hopeful approach would be to have the student work on concrete problems under guidance, in an active and constructive environment." [72]

An account of the need for a reorientation in software engineering, which is in close agreement with the dominant attitude in the Copenhagen Tradition of computer science, has been given by Christiane Floyd [20], who through her empirical studies has also become critical of the strong belief in principles and rules in software development that is so prevalent in the academic world [19]. A group of researchers, the so-called Mars group, originally from Aarhus University,

has reached similar conclusions through their empirical studies [3].

5. Concluding remarks.

The Copenhagen Tradition of computer science is characterized, first and foremost, by maintaining a close connection with applications and other fields of knowledge and, in particular, the understanding of programming and system development as a human activity. In our computer science education comprehensive project activity is an integral part of the curriculum, thus presenting theory as an aspect of realistic solutions known primarily through personal experience obtained under active guidance from researchers and more advanced students. Intuition and talent for theory building during programming are crucial for programmers and computer scientists, and we are trying to train this by working with examples, solving problems, participating in project work, and studying the work of others. Inherent in the approach is that dogmatic ideas are constantly challenged and people are being supported in revising their views in light of new insight or due to changed circumstances. We feel, that having the students gain such experiences gives them a basis for powerful and responsible professional conduct.

However, according to Naur our approach to computer science can be difficult to maintain. Strong forces within both academia and industry are behind a widening gap between the more academic, "pure" computer science oriented study of programming and the world of practical programming. Though highly problematic, this gap can be expected to persist indefinitely. This is closely treated in [60], drawing on similar historical experiences within natural languages and mathematics. In a public debate on computer science in Denmark in 1983 the question of the desirability of such a split between "pure" and "applied" computer science was raised. Naur strongly advocated to oppose this splitting. Let us conclude this paper by a quotation from the debate, drawing attention to the fact that our approach in spite of its obvious qualities will require continued inner and outer struggles [71]:

> "One will always be faced with the difficulty of deciding whether what one is doing is scientifically defensible, if it is valuable enough. This kind of nagging doubt is unknown in the pure subjects. There, researchers adopt their own basis of evaluation independently of demands made by the complicated and unclean reality. However, at DIKU we have hitherto been able to maintain our applications-oriented line, and we have attracted many students who have been able to use what we try to teach them." [*Translated from Danish.*]

Acknowledgments.

It is always difficult to write about something that is close to one's heart and of which one is a part - it becomes somehow invisible. Therefore we want to thank Christian Andersen, Nils Andersen, Cathrine Blicher, Christian Gram, Keld Helsgaun, Jesper Holck, Peter Johansen, Neil D. Jones, Eric Jul, Anker Helms Jørgensen, Finn Kensing, Jakob Krarup, Peter Møller-Nielsen, Flemming Sejergaard Olsen, Lissi Olsen, and Ernest M. Stokely for reading the manuscript and helping us make it real. Special thanks are due to Christiane Floyd, Saul Gorn, H. B. Hansen, Elin Rønby Pedersen, and A. P. Ravn who contributed with in-depth criticism and suggestions for

improvement. We thank Birthe Hougaard and Else Høyrup for library assistance, Jens A. Lassen for technical help and Margaret Malone for translating the Danish quotes and improving the language.

References.

1. J. Damgaard Andersen, J. Hilden, G. Leunbach, P. Naur, J. Born Rasmussen, T. Warnich-Hansen, and P. Winkel, *Retningslinier for universitetskurser i datalogi som hjælpefag. Vejledning for lærere ved tilrettelægning af fagorienterbar undervisning (Guidelines for university courses in datalogy as an auxiliary subject. Handbook for teachers in the organization of subject-oriented teaching),* Københavns Universitet, Datalogisk kursusgruppe under edb-udvalget, Datalogisk Institut, 1974, 95 p.

2. P. Lindblad Andersen, P. Møller-Nielsen, and E. Sveinsdottir, *Datalogi ved Københavns Universitet*, Meddelelser fra Københavns Universitet, Vol. 17, no. 19, May 25, 1969.

3. N. E. Andersen, F. Kensing, M. Lassen, J. Lundin, L. Mathiassen, A. Munck-Madsen, and P. Sörgård, *Professionel systemudvikling*, Teknisk Forlag, København, 1986.

4. P. Brinch Hansen, *The nucleus of a multiprogramming system*, Comm. ACM, Vol. 13, no. 4, 1970, pp. 238-241, 250.

5. P. Brinch Hansen and Roger House, *The Cobol Compiler for the Siemens 3003*, BIT 6, 1966, pp.1-23.

6. S. Bundgaard, *Hvordan edb kom til Aarhus Universitet*, doc. RECAU-78-86-R, Sept. 1978.

7. *Curriculum 68, Recommendations for Academic Programs in Computer Science*, Comm. ACM, Vol. 11, no. 3, 1968, pp. 151-197.

8. T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, 1979.

9. P. J. Denning, *Paradigms Crossed*, Editorial, Comm. ACM, Vol. 30, no. 10, 1987, pp. 808-809.

10. P. J. Denning, *A hard look at Structured Programming*. In *Structured Programming*, Infotech State of the Art Report, 1976, pp. 183-202.

11. E. W. Dijkstra, *Go To Statements Considered Harmful*, Comm. ACM, Vol. 11, no. 3, 1968.

12. E. W. Dijkstra, *The Humble Programmer*, Comm. ACM, Vol. 15, no. 10, 1972.

13. E. W. Dijkstra, *Notes on Structured Programming*. In O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare: *Structured Programming*, Academic Press, 1972 or later, pp. 1-82.

14. DIKU-blad 3.9, *Immatrikulerede datalogistuderende ved Københavns Universitet oktober 1987*, Datalogisk Institut, Københavns Universitet, March 1988, 11 p.

15. O. Dopping, *Suggestion for teaching of administrative data processing at Swedish universities*, BIT, Vol. 5, 1965, pp. 73-84.

16. H. Dreyfus and S. Dreyfus, *Mind over Machine*, The Free Press, N.Y., 1986.

17. P. Ehn and M. Kyng, *A Tool Perspective on Design of Interactive Computer Support for Skilled Workers*, Proc. 7th Scandinavian Research Seminar on Systemeering, Helsinki, 1984.

18. L. Fein (org.), *Panel on University Education in Information Processing*, IFIP Congress 1962, pp. 763-765.

19. C. Floyd, *A Comparative Evaluation of System Development Methods*. In T. W. Olle, H. G. Sol, A. A. Verrijn-Stuart (eds.): *Information Systems Design Methodologies*: *Improving the Practice*, North-Holland, Amsterdam, 1986.

20. C. Floyd, *Outline of a Paradigm Change in Software Engineering*. In G. Bjerknes, P. Ehn and M. Kyng (eds.): *Computers and Democracy*, Avebury, Gower Publishing Co., England, 1987.

21. G. Forsythe, *A University's Educational Program in Computer Science*, Comm. ACM, Vol. 10, no. 1, 1967.

22. S. Gorn, *The Computer and Information Science: A New Basic Discipline*, SIAM Review, Vol. 5, no. 2, l963, pp. 150-155.

23. C. Gram, O. Hestvik, H. Isaksson, P. T. Jacobsen, J. Jensen, P. Naur, B. S. Petersen, and B. Svejgaard, *Gier - A Danish Computer of Medium Size*, IEEE Transactions on Electronic Computers, no. 5, 1963, pp. 629-650.

24. D. Gries, *The Science of Programming*, Springer-Verlag, 1981, p. 297.

25. I. Gryndahl, T. Møller, K. Hansen, F. Sejergaard Olsen, O. Caprani, A. P. Ravn, and S. Jürs, *Utopia II: Situationsrapport for Datalogisk Institut ved Københavns Universitet*, Copenhagen University, summer 1970, 33 p.

26. M. A. Jackson, *Principles of Program Design*, Academic Press, 1975.

27. M. A. Jackson, *System Development*, Prentice Hall International, 1983.

28. P. Johansen, *Datalogi*, Københavns Universitet 1479-1979, Vol. XII, København, 1983, pp. 201-211.

29. *"Johnsen rapporten", Betænkning om edb-undervisning i det offentlige uddannelsessystem*, Betænkning 666, Undervisningsministeriet, 1972.

30. A. Helms Jørgensen, *On the Psychology of Prototyping*. In R. Budde, Kuhlenkamp, and L. Mathiassen (eds.): *Approaches to Prototyping*, Proceedings of a working conference in Namur, Oct. 1983, Springer, N.Y., 1984, pp. 278-289.

31. *Kandidater i matematik-, fysik- og kemifagene: Hvor gik de hen?*, Fysisk Institut, Århus Universitet, 1987, 112 p.

32. D. E. Knuth, *Structured Programming With go to Statements*, Computing Surveys, Vol.6, no. 4, 1974, pp. 261-301. Also in R. T. Yeh (ed.): *Current Trends in Programming Methodology*, Vol. I, Software Specification and Design, Prentice Hall, 1977, pp.140-194.

33. S. Lammers, *Programmers at Work - Interviews*, Charles Simonyi, pp. 6-24, Microsoft Press, 1986.

34. Aa. Melbye and P. Sveistrup, *The need for education and research in administrative data processing*, BIT, Vol. 2, l962, pp. 35-44.

35. P. Naur (ed.), J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger, *Report on the Algorithmic Language Algol 60,* Comm. ACM, Vol. 3, no. 6, 1960, pp. 299-314.

36. P. Naur, *The Design of the Gier Algol Compiler,* BIT, Vol. 3, 1963, part I: pp. 124-140, part II: pp. 145-166.

37. P. Naur, *Forslag og ønsker vedrørende datalogien ved Københavns universitet (Proposals and wishes as to datalogy at Copenhagen University),* unpublished note, January 1969, 5 p.

38. P. Naur, *Go to statements and good Algol style,* BIT, Vol. 3., 1963, pp. 204-205.

39. P. Naur, *The Place of Programming in a World of Problems, Tools and People,* Proc. IFIP Congress 65, Vol. I, 1965, pp. 195-199.

40. P. Naur, *Proof of algorithms by general snapshots,* BIT, Vol. 6, 1966, pp. 310-316.

41. P. Naur, *The Science of Datalogy*. Letter to the editor. Comm. ACM, Vol. 9, no. 7, 1966, p. 485.

42. P. Naur, *Program Translation Viewed as a General Data Processing Problem,* Comm. ACM, Vol. 9, 1966, pp. 176-179.

43. P. Naur, *Plan for et kursus i datalogi og datamatik (Plan for a course of datalogy and datamatics),* A/S Regnecentralen, København, March 1966, 64 p. See Computing Reviews for English description, CR 16,098.

44. P. Naur, *Datalogi og datamatik og deres placering i uddannelsen (Datalogy and datamatics and their place in education),* Magisterbladet no. 10, May 15, 1966.

45. P. Naur, *Datamaskinerne og samfundet (Computers and society),* Munksgaards Forlag, 1967, 104 p.

46. P. Naur, *Demokrati i datamatiseringens tidsalder (Democracy in the age of computerization),* Kriterium, Vol. 3, no. 5, 1968, pp. 31-32.

47. P. Naur, *Datalogy, The Science of Data and Data Processes and its Place in Education,* Proc. IFIP Congress 68, North-Holland, 1969, pp. 1383-1387.

48. P. Naur, *Project Activity in Computer Science Education,* Lezione "Leonardo Fibonacci 1969", Pisa, April 1970, 13 p.

49. P. Naur, *A course of datalogy for radio and TV,* NordDATA-69, Part 3, 1969, pp. 53-60.

50. P. Naur and B. Randell (eds.), *Software engineering,* Nato Science Committee, Brussels, 1969, 231 p. Reprinted in Naur, P., B. Randell and J.N. Buxton (eds.): *Software Engineering: Concepts and Techniques,* Petrocelli/Charter, New York, 1976.

51. P. Naur, *Programming by Action Clusters*, BIT, Vol. 9, 1969, pp. 250-258.

52. P. Naur, *Datalogi i skolen (Datalogy in the school)*, Notabene, Gjellerups Forlag, Vol. 6,

no. 2, December 1970, pp. 3-6.

53. P. Naur, *Planer og ideer for et datalogisk institut ved Københavns Universitet (Plans and ideas for an institute of datalogy at Copenhagen University)*, Studentlitteratur, Lund, 1970, 89 p.

54. P. Naur, *Problemformulering - edb-projektets grobund (Problem formulation - the fertile soil of the EDP project)*, Data, no. 1, 1971, pp. 46-48.

55. P. Naur, *An Experiment on Program Development*, BIT, Vol. 12, no. 3, 1972, pp. 347-365.

56. P. Naur, *Programming languages - status and trends*, Data, no. 6, 1972, pp. 36-38.

57. P. Naur, *Concise Survey of Computer Methods*, Studentlitteratur, Lund, 1974, 397 p.

58. P. Naur, *What happens during Program Development - an Experiment*. In Lundeberg and Bubenko (eds.): *Systemeering 75*, Studentlitteratur, 1975, pp. 269-289.

59. P. Naur, *Trends in Computer Science Education.* In Lecture Notes in Computer Science 26, Springer-Verlag, 1975, pp. 85-93.

60. P. Naur, *Programming Languages, Natural Languages, and Mathematics*, Comm. ACM, Vol. 18, no. 12, 1975, pp. 676-683.

61. P. Naur, *Control Record Driven Processing.* In *Structured Programming*, The Infotech International State of the Art Report, 1976, pp. 309-322. Also in R. T. Yeh (ed.): *Current Trends in Programming Methodology.* Vol. I, Software Specification and Design, Prentice Hall, 1977, pp. 220-232.

62. P. Naur, *Software Reliability*. In *Reliable Software*, Vol. 2: Invited Papers, Infotech State of the Art Report, Maidenhead, England, 1977, pp. 243-251.

63. P. Naur, *Review 33.686: John Backus: Can programming be liberated from the von Neumann style? A functional style and its algebra of programs*, Computing Reviews, Vol. 20, no. 11, 1978, p. 445.

64. P. Naur, *Diminishing returns of user programming*. In Infotech State of the Art Report: *Future programming*, Vol. 2, 1978, pp. 143-149.

65. P. Naur, *Impressions of the early days of programming*, BIT, Vol. 20, 1980, pp. 414-425.

66. P. Naur, *An empirical approach to program analysis and construction.* In *Systems Architecture*, Proc. 6th ACM European Regional Conference, ICS 81, 1981, pp. 265-272.

67. P. Naur, *The European side of the last phase of the development of ALGOL 60*. In R. L. Wexelblat (ed.): *Proc. History of Programming Languages Conference*, Los Angeles 1978, Academic Press, New York, 1981, pp. 92-139, 147-161.

68. P. Naur, *Prospects for the programming methodologies*, Infotech State of the Art Report, Ser. 9, no. 6, System Design, 1981, pp. 293-300.

69. P. Naur, *Formalization in Program Development*, BIT, Vol. 22, 1982, pp. 437-453.

70. P. Naur, *Program development studies based on diaries*. In T. R. G. Green, S. J. Payne, G. C. van der Veer (eds.): *Psychology of Computer Use*, Academic Press, London, 1983,

pp.159-170.

71. P. Naur, *Datalogiens veje og vildveje (The ways and blind ways of Datalogy)*, Weekendavisen, December 9, 1983.

72. P. Naur, *Programming as Theory Building*, Microprocessing and Microprogramming 15, 1985, pp. 253-261.

73. P. Naur, *Intuition in Software Development*. In H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher (eds.): *Formal Methods and Software Development*, Vol. 2: Colloquium on Software Engineering, Lecture Notes in Computer Science 186, Springer-Verlag, Berlin, 1985, pp. 60-79.

74. P. Naur, *Review 8502-0062 of D. Michie: Machine intelligence and related topics*, Computing Reviews, Vol. 26, no. 2, 1985, pp. 101-104.

75. P. Naur, *Thinking and Turing's Test*, BIT, Vol. 26, 1986, pp. 175-187.

76. P. Naur, *Programming Studied from Case Activity Records*, Invited lecture to Fifth Symposium on Empirical Foundations of Information and Software Science, November 1987, Risø National Laboratory. To be published in *Empirical Foundations of Information and Software Science*, Plenum Press, N.Y.

77. P. Naur, *Programmeringssprog er ikke sprog (Programming languages are not languages)*, Mål og Mæle, Vol. 12, no. 2, 1988, Gads Boghandel, København, pp. 24-31.

78. D. L. Parnas, *On the Design and Development of Program Families*, IEEE Trans. Software Engineering, March 1976, pp. 1-9. Also in D. Gries (ed.): *Programming Methodology*, Springer Verlag, 1978, pp. 360-361.

79. Planudvalget for DIKU, *Forslag til afhjælpning af problemer ved Datalogisk Institut*, Det naturvidenskabelige Fakultet, Københavns Universitet, February 1983.

80. G. Ryle, *The Concept of Mind*, Peregine Book, Penguin Books Ltd., 1963. First published by Hutchinson, 1949.

81. E. Sveinsdottir og E. Frøkjær, *Datalogi og informatik ved Københavns Universitet*, Uddannelse, Vol. 17, no. 1, Undervisningsministeriet 1984.

82. P. Sveistrup, P. Naur, H. B. Hansen, and C. Gram (eds.), *Niels Ivar Bech - en epoke i edb-udviklingen i Danmark*, Data, 1976.

83. *An Undergraduate Program in Computer Science - Preliminary Recommandations*, Report from the ACM Curriculum Committee on Computer Science, Comm. ACM, Vol. 8, no. 9, 1965.

84. T. Winograd and F. Flores, *Understanding Computers and Cognition - A new foundation for design*, Ablex Publishing Co., 1986.

85. N. Wirth, *Program Development by Stepwise Refinement*, Comm. ACM, Vol. 14, April 1971, pp. 221-227.

86. E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Programming and Systems Design*, Prentice-Hall, 1979.

87. E. Yourdon, *Managing the System Life Cycle, a software development methodology overview*, Yourdon Press, N.Y., 1983.