

# The internal state of computing science in historical perspective

NICO F. BENSCHOP

Philips Research, Eindhoven, The Netherlands. Sept. 1973

## Abstract

The current internal state of computing science is estimated from the research reported at the International Computing Symposium 1973 (Davos). The controversy between conservative and progressive wings in higher level languages is described and taken to represent a similar situation throughout computing science. Historic review and analogy are used as a method to put the present state ('software crisis') in perspective, and to locate the bottleneck impeding fundamental progress. This yields insight into a more desirable next-state, and the input required to get there. As an inspiring example is taken physics in its emergence from the Middle Ages. Basic concepts for progress are recognized in modern algebra, especially *closed system* and *internal state*, thus *memory* and *non commutative* algebra. These concepts provide a link between soft- and hardware c.q. function and construction of sequential behaviour, depending on progress made in a general structural and quantitative theory of *finite associative* systems. The spectrum of papers at *ICS73* is analysed.

## Contents:

Abstract

1. The 'software crisis' controversy
2. Analogy, iteration and context-free tricks
3. The bottleneck
4. Specialisation and coupling
5. Emphasis of *ICS73* and conclusion.

## 1 The 'software crisis' controversy

The International Computing Symposium *ICS73*, 4-7 September 1973, took place in the Congress Center of Davos in Switzerland. It was organized by the European Chapters of the Association for Computing Machinery (*ACM*), especially the Swiss Chapter.

Apart from discussing a few very interesting papers, I will try to perceive and analyse the conference as a whole. It represented a main stream of present-day computing science, judging by its size. The enormous variety offered forced me to take distance in an effort to put it in perspective, possibly inspired by the beautiful environment of the conference.

The scope of the symposium ranged over the whole field of computing science, however, with a strong emphasis on applications, as may be appreciated from the following overview of titles of the keynote address, the 10 invited papers and 24 sessions (representing 76

contributed papers and 2 panel discussions). The labels T, A, P, E (Task, Analysis, Programming, Execution) refer to a classification explained later.

**Keynote Address:**

A. Ralston (Amherst): The future of higher level languages (in teaching). (T)

**Invited Papers:**

V. Strassen (Zürich): Evaluation of rational functions. (A)

R. Needham (Cambridge): Protection: current research in operating systems (E)

I. de Lotto (Pavia): Sparse matrix techniques in computer aided design. (T)

D. Edwards (Manchester): Hardware innovation and computer design. (E)

N. Wirth (Zürich): From programming techniques to programming methods (A)

P. Deussen (Karlsruhe): Description of processes. (A)

G. Dahlquist (Stockholm): Problems in numerical treatment of stiff *DE*'s. (T)

H. Kazmierczak (Karlsruhe): Problems in automatic pattern recognition. (T)

C. Toulet (Paris): Real-time systems in administrative data processing. (T)

M. Wilkes (Cambridge): Past, present and future in the computer world. (E)

Table 1: Sessions of ICS-73, Davos, 4–7 sept 1973

Date	Sessions 1 and 3	Sessions 2 and 4
4 Sept pm	Operating Systems (E) Theory of Computation (A)	Data Bases (A) Priorities in Education (T)
5 Sept am	Computer Networks (E) Programming Languages (P)	Computer Aided Design (T) Non-numerical Applications (T)
5 Sept pm	Programming methods (P) Computer Design (E)	Concurrent Proc. Resource Alloc. (E) Comp. Aided Instruction (T)
6 Sept am	Info Storage & Retrieval (A) On-line Applications (T)	Simulation Syst.Meas. (E) Pattern Recogn. Appl. (T)
6 Sept pm	Compilers (P) Applied Mathematics (T)	Micro Programming (P) Large Scale Softw. Systems - panel (E)
7 Sept am	Medical Applications (T) Pattern Recogn. Methods (T)	Comp. Graphics Math. Appl. (T) Numerical Mathematics (T)

The present-day controversy between conservative (pragmatic, realistic, subject to context) and progressive (dogmatic, theoretical, context-free) wings, most visible in the area of high level languages, was clearly exposed by dr. A. Ralston, president of the *ACM*, in his keynote address on "The future of higher level languages (in teaching)" and N. Wirth, professor at ETH Zürich and originator of the 'Pascal' programming language, in his invited paper "From programming techniques to programming methods".

It is worthwhile to study this controversy a bit more closely, because the stronger it persists and the more people are involved, the more likely it relates to opposite sides of the same coin, long overdue for recombination.

Dr. Ralston asserted : "Despite numerous competitors and despite the efforts of some very distinguished people, FORTRAN - almost 19 years old - continues to be the predominant language for scientific applications and for teaching in the US. And the situation in Europe

is not very different". After raising some questions why this should be so, he admits the 25 years that electronic computers exist are marked by numerous major revolutions, and he expects more to come. "Yet my thesis here is founded on an assumption whose truth I believe, although many will not, that *there will be no more revolutions in higher level languages*, at least in that category usually called general purpose higher level languages. In other words, the state of the world of general purpose higher level language usage is very stable, with a high degree of inertia, and for better or worse there is little or nothing on the software or hardware horizon which will greatly or quickly change that state".

In the section on "potential for change" he discounts the possibility of clean programming. "There is however one thing on the horizon with the potential to upset the dominance of the *Fortran-Cobol* system. This is the very interesting and, from a theoretical point of view, crucial research in *proving the correctness of programs*, which is now being actively pursued at a number of institutions in Europe and the US. Conceivably, just conceivably, there could be a breakthrough in this area which would so clearly illuminate the task of programming, that almost everyone would see the light and agree to a change of methodology, and - because of this - a change of language. I wish such a breakthrough were likely, but I don't believe it is, and I will therefore discount this possibility in what follows."

After thus professing that, as far as he was concerned, the pioneering days are over, he spelled out the consequences of his assumption that *FOTRAN* is and will remain dominant and practically immovable, so that progress only occurs from within by expansion.

Professor N. Wirth (*ETH* Zürich) began his paper

"From programming techniques to programming methods" in rhyme :

"He coded in *FOTRAN* like hell,  
wrote programs with whistles and bell.  
Now feels that his tool  
has made him a fool,  
but cannot get rid of its spell."

" This piece of modern poetry characterizes the stage of programming, the predicament of the human actors on it, and singles out one of its harmful immovables... Tricks were necessary at that time (the first decade of computers up to the early sixties) simply because machines were built with limitations imposed by a technology in its early development stage, and because even problems that would nowadays be termed simple, could not be handled in a straight forward way. It was the programmers very task to push computers to their limits by whatever means available. As computers grew more powerful the elimination of deficiencies, errors and blunders ('debugging') became the overwhelming problem."

One of the reasons that *the remedy: a structured language* like *ALGOL-60*, was not very successful is that "it appeared on the scene when the relevance of *structure* had not yet been widely recognized, and its restrictiveness against the use of clever tricks was considered a handicap and a deficiency. The law of the 'Wild West of Programming' was still in too high esteem! Another reason was the growing success of Fortran which, due to an unwritten law of inertia, later became its biggest drawback, and still plagues the programming scene today."

When the complexity of programs and machines continued to grow, ... "it was gradually recognized that the true challenge does not consist in pushing computers to their limits by saving bits and micro-seconds, but in being able to organize large and complex programs,

and assuring that they specify a process that for all admitted inputs produces correct results. In short, it became clear that any amount of 'efficiency' is worthless without *reliability*. We must recognize the strong and undeniable influence that our language exerts on our ways of thinking, and in fact defines and delimits the abstract space in which we can formulate - give form to - our thoughts. But now the term *structured programming* has been coined, and it is finally achieving what the term 'structured language' was unable to suggest."

He then stressed that programming of a task should in the first place be a systematic analysis and decomposition of it, carefully restricting ourselves to what we can manage intellectually, thus: being aware of the alternatives to - and implications of - each decision we take in the design process.

These two opposing attitudes, most pronounced in the context of programming languages as detailed above, pervade other areas of computing science as well. The conservative one is not unhappy enough with the admittedly far from ideal status-quo to do something about it, and content with computers doing the rest, for better or worse... The progressive wing, on the other hand, experiences the present state as a disastrous 'software crisis', and foresees in the near future a flood of information processing if the brooms of the sourcerer's apprentice are not brought under control.

Paradoxically, the slogan of the progressive wing is "safety first", aware of the power of a computer science comparable to physics in the Middle Ages, possibly on its way to become a respectable quantitative science, with monotone- and conservation laws, necessary for the concise expression and design of sequential (thus *dynamic*) behaviour.

## 2 Analogy, iteration and context-free tricks

The basis and substance of understanding are *analogy* and *iteration*, representing its qualitative (inductive) and quantitative (deductive) aspects respectively. Analogy transfers us between contexts, and also separates detail from essence (abstraction) - while iteration unfolds structure within one context.

One may substitute *morphism* for 'con-sistent an con-sequent symbolic representation', or *interpretation* for 'analogy', and *experience* or routine for 'iteration'. The value of intuition obtained by analogy increases with experience. And conversely, the quality and stability of the substance of experience depends on that of its basis. Clearly the progressives feel that the basis is still too soft and amorph to grant mass iteration.

However, one consolation is that the bug-generating process is self regulatory. In the Middle Ages, before the Arabic notation and decimal point were re-invented (somehow lost since the base 60 notation of the Sumerians, 2500 BC), one could not possibly operate efficiently and reliably with many, big and accurate numbers .

To quote E.T. Bell (in *Development of Mathematics*, p34): "The more honor then to the ancients, who persevered through jungles of words to obtain what the moderns reach with a few almost mechanical strokes of the pen."

This suggests that it is profitable to pay extra attention to the basis. After a good but short-lived start of inductive and deductive mathematics and science, it took some fourteen centuries to resolve the multitude of context sensitive epi-circles describing our planetary system (Ptolemy), by such elegant *and* efficient principles as:

- choosing an efficient center of description for maximal symmetry [Copernicus],
- defining the proper primitives (by generalizing circles to quadratic loci: ellipses) [Kepler]
- and their *generative dynamic* description (up to iteration) [Newton]
- by the concepts of *state* and *next-state* function (differential equation)
  - in a monotone one-way universal parameter (time),
- while coupling state (place and impulse) to input (time)
  - via the intermediate concept of *force* (second derivative of place).

The nature of these principles, however, is typically that of clever coding tricks, growing in stature proportional to the size of the context in which they appear to be valid. That is: the iteration they would allow, in place and time, before a contradiction is detected (subject to memory).

Some tricks grew via techniques, to methods and laws, which are context-free tricks in optima forma. No doubt such principles are found via analogy, conjecture and experiment, in a dynamic balance of give and take. Experience (data) is reduced to a near-trivial *core* of efficient elegance, by recognizing repeated similarity, leaving to substance nothing but iteration (time, number). But this is the very idea of automation.

Conversely: clean structure, thus a minimal core of concepts, allows efficient (re-)generation or representation of the observed, and similar, data.

The essential methods of mathematics, of science, and of automation coincide, only differing in depth and generality. Find the concepts allowing precise expression of these axioms (assumptions, restrictions) which lead in a direct and efficient manner to correct results.

### 3 The bottleneck

To appreciate the spectrum of activities reported at ICS73, as reflected in the 35 titles of sessions and invited papers, some classification, ordering and quantification is helpful. There are two viewpoints from which to describe a computer, or any component:

1. Its *external* description or I/O *function* as a component in some application;
2. Its *internal* description or *construction*, as a network of simpler components, each given by its function and its coupling to other components.

The *sequence restrictions* in the output with respect to the input, characterizing the external description, translate to a particular interconnection pattern, or layout, i.e. *spacial restrictions* between the component functions forming the network, and vice versa. The total I/O function is as it were distributed over a network of simpler functions. The spacial coupling pattern depends of course on the choice of admitted component functions that serve as a-tomic in-divisible 'prime' actions for that level of internal description.

This choice of prime actions, and the transition from sequential to spacial restrictions, seem to be the main issues that require further analysis. Ordering, equivalence and a consistent measure of complexity of sequential (next-state) functions form are of essence.

Regarding subjects in computing science, consider the following four-fold classification as a working hypothesis. Applications (task T) and their Analysis (A) with programming and execution in mind, are external to the computer. Programming (P) and machine design + operating system (execution E) with tasks and their analysis in mind, are internal to

it. They may be thought of as following and influencing each other in a cyclic manner, as depicted in the next figure:

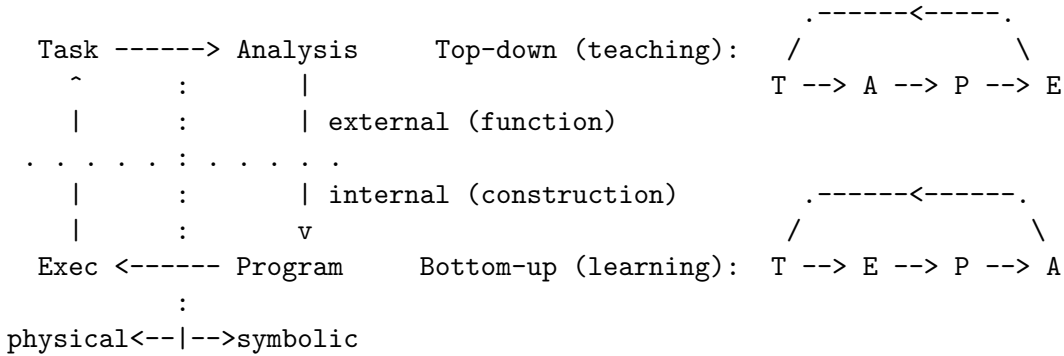


Fig.1 : the four-fold T,A,P,E classification.

Eventually each activity influences all, including itself. The effect of completing a cycle is to increase experience and the quality of each particular activity. This aspect is not represented in the cyclic diagram, which is a plane projection of a learning resp. teaching spiral: left- or right- turning respectively, being each others inverse, or equivalently: each others third iteration (in the 4-cycle).

"Task" and "Program" are *networks* of tasks and programs, which are transferred into each other by the *methods* of "Analysis" and "Execution".

Historically, since the late 1940's, the influence was inverse to the indicated direction. Increasingly complex tasks motivated the construction of electronic computers (Exec) which, after considerable symbiotic expansion of both, soon led to the need for programming (micro- and macro-, sequential- and parrallel-).

Trying to oversee tasks and their programmed execution in turn revealed gaps in mathematical concepts and analytical methods to model tasks, and to design machines. This bottom-up process is typically that of additive generation ('Peano') which, when approaching a certain threshold of complexity, urges a (top-down) review in order to obtain a more efficient representation by multiplicative ('prime') factorization.

As tasks and programs turn around faster and faster, it is inevitable that methods of analysis and execution come closer together, in order to keep things under control. The concept of *internal state*, typical for the fixed format of hardware, and useful to fix semantics, is equivalent in integrated form to associativity resp. transitivity in algebra ('inner translations' - Cayley 1878). This lead at the end of the nineteenth century to molecular thermo dynamics and the concept of *entropy*. "Internal state" was introduced by Turing (1936) to start computing science in the first approach to define what is computable, while 'entropy' was used by Shannon (1948) for a statistical coding theory of communication.

The multitude of types of automata that mushroomed from this concept of *internal state*, comparable to the large variety of special associative systems (semigroups) under study in pure mathematics, would paralyze even the most courageous, and indeed has done much harm, in the same way as the addiction to 'extra features' in some computer languages.

The best way to proceed seems to return to first principles, of some hundred years ago, however with a better specified goal in mind. Design decisions regarding:

- soft- vs. hardware
- free vs. fixed format
- wild vs. regular logic
- control- vs. operation unit
- sequential- vs. parallel processing
- algorithmic computation vs. table lookup

are at the heart of programming and computing.

They all relate to the role of memory in computation, representing a trade-off, or coupling, between space and time, which is intuitively felt rather than quantitatively measured.

This situation seems to be imaged in the separation between combinational and sequential analysis in mathematics. Slowly the *lack of coupling* between commutative idempotent analysis (such as combinational logic, Boolean algebra and lattice theory) on the one hand, an non-commutative sequential, but periodic, analysis (like group theory) on the other, emerges as the *critical section* impeding real progress in understanding and quantifying sequential behaviour (viz. with memory) in space and time.

This description of the bottleneck points to a solution of the above dichotomy: release the restrictions *commutative*, *idempotent* and *periodic* (existence of an inverse: time-symmetry) separating the two approaches, in order to develop a discrete and finite analysis based on their common property of associativity, emphasizing structure and quantification.

In physics the concepts of *closed system* (invariance of mass + energy) and *continuity* have done good service, so why should they not, in the form of algebraic closure (viz. invariance) and associativity, do the same for informatics?

When assuming closure and associativity only, the temptation is strong to narrow down the search by increasing the restrictions, in order to speed up progress. Resisting this urge avoids the task of recombining ways which parted too soon and diverged too far.

A balance between width and depth is necessary for stable growth. The restriction to associative closure is already a very strong one, comparable to that of continuity of functions, increasing the chance of obtaining sharp structural results, while on the other hand it leaves sufficient freedom for the great variety of purposes in computing science.

However, viewing the bottleneck from this angle, one may appreciate the considerable obstacles to be cleared or avoided. The theory of associative systems, better known in mathematics under the anachronistic name of *semi-groups*, is hardly 50 years old (Schuskevitch 1928: the detailed structure of finite simple semigroups). The need for unification of the bits and pieces has been much less than in computing science today (1973). Still, such unification may be the shortest way to the definition of those restrictions that yield practical tools for designing and computing in a variety of contexts.

A structural and quantitative representation theory of associative systems, with the concepts of internal state coding, finite machines and constructive algorithms in mind, will probably not be developed by pure mathematicians. Practical interest may spoil their field, which they hope "will never be of any use" (quoting number theorist Hardy) in order to keep it clean. But they are already much further, exploring by habit the infinite, leaving to others (engineers) the unglorious and difficult fine structure in the limitations

of the finite. As usual, they are some hundred years ahead of time, assuring - in the subjective sense of 'never'- their claim of uselessness, and with it their essential tool of freedom and flexibility of thought.

## 4 Specialisation and Coupling

The time and energy that separate a task from its execution, have a mental term in analysis and programming, and a physical one in execution by computer hardware, with a negligible mental term for output interpretation in case of good task analysis.

There are two kinds of efficiency: one of concise and clear representation (up to iteration), necessary for understanding and communication, and the other of execution. These kinds of efficiency are probably not independent of each other. Some feel they are opposed, others are convinced they may, or even must necessarily, coincide. One is inclined to expect the complexity of a system, and that of its defining relations, to be inversely proportional, in order to prevent bugs - and to keep it working. Consider for instance such a complex system as (inanimate) nature. It has basic relations, say between mass - force - energy - entropy - place and time, which are of very simple type: quadratic, first or second order, symmetric, invariant or monotone. They define nature's next-state function, which delimits its freedom to act, while the (again quadratic) principle of least action - or maximal efficiency - appears to be its only choice, as far as we can observe.

Although we, and our problems, are part of nature, our inefficiency seems to defy its basic laws. The reason for this anomaly seems to be that, while inanimate nature is its own memory - in the form of inertia - hence cannot be inconsistent, we suffer from a separation of our motoric (external) and memory (internal) functions. So we are left with the burden of establishing a consistent correspondence between them.

The advantage of this functional separation is a gain in flexibility: we can we can imagine and simulate, assuming a proper model of the problem at hand, making us independent of the one-way and one-speed which time forces upon the Uni-verse. This allows us to consider the consequences of the alternatives given by our imagination, and to choose our way accordingly, assuming a proper criterion for comparison. These two assumptions, of model and criterion (analogy and quantification), represent the consistent correspondence to be established.

But the gain in flexibility, due to the separation of inert motoric and almost mass-less memory (thinking) functions, is reduced by the effort of finding adequate symbolisms to link them. As usual, the gain of specialisation is diminished, hopefully not eliminated, by the *cost of coupling*.

This problem of coupling is also characteristic for the difficulties of (de)composition in associative system theory. It is similar to that of the *carry* in arithmetic, which Babbage (of the Calculating Engine, 1820) was most proud to have solved to his satisfaction.

Disjoint decomposition of (commutative) systems is relatively simple, but the essence: *joint* composition of (inter-dependent) systems is still at large. It is probably so close at hand and simple, as to escape attention - like the air through which we communicate. The solution to coupling *has* to be very simple to be of any reliable use in large information systems. After all 'com-puting' means 'putting together', hence coupling...

## 5 Emphasis of *ICS73* and conclusion

Returning to *ICS73*, let us apply the T-A-P-E classification (Task, Analysis, Programming, Execution) to the 35 subjects at the conference. The content, rather than the title of a session or paper is used to choose the proper class from the above sketched point of view. For instance, mathematical tools of a very special nature, like stiff differential equations, sparse matrices in CAD, numerical analysis, fall under applications ('Task'), since they are too directly linked to specific applications to fall under Analysis in the above sense of overseeing tasks and their programmed execution.

New areas of research, like data-base structure, information storage and retrieval (unless of a simple applicational nature) and theory of computation, are classified under Analysis. Areas like protection, operating systems and resource allocation, fall under 'Execution' if they relate to problems created by execution, rather than by applications per sé, although they may signal phenomena badly needing basic analytical tools.

Like any classification, this is a subjective one, obtaining its value from the insight provided by its point of view. The presentation of papers was organized as follows. As a rule, every morning saw two invited papers, followed by four parallel sessions - each of three papers in sequence. And each afternoon: one invited paper, preceeding four parallel sessions of four papers in sequence. Hence, each invited paper and panel discussion will count for 4 papers (units) corresponding to the number of papers they replace in the program.

This amounts to a total of  $11.4 + 3(3+4).4 = 128$  units.

The initials T,A,P,E encode the classes and are used to label the subjects, as done in the table of section 1. The next distribution then appears:

Table 2: Quantified emphases of *ICS73*

	Task	Analysis	Program	Execution	total
Invited talks	20	8	4	12	44
Session papers	36	11	15	22	84
total	56	19	19	34	128
	44%	15%	15%	26%	100%

The outside of a computer (task + analysis) attract some 60% of attention, as compared to 40% for its inside (programming + execution).

On the other hand, the 70% for task + execution better exposes the emphasis of *ICS73*, with a meager 30% for analysis + programming to observe the hurry. In an industrial environment this would not be surprising. However, for a symposium of research at universities and other institutions, this division of attention seems to emphasize, rather than to solve the present 'software crisis'. The call of Wirth for "safety first" needs not only more attention but active support, to balance Ralston's rally for expansion from within.

With respect to the language controversy, an outsider may recognize the efficient conciseness of the English language, at the cost of a high dependence on context, as mirrored in *FORTTRAN*, versus the block-structure of the German or Dutch language, requiring a strong memory to recall the first half of a word split for bracketing purposes, as imaged in *ALGOL*. The identification of their greatest common denominator could serve to express most efficiently the characteristics and differences between them, in order to reach the next level of understanding context and history, necessary for balanced progress.

The following papers are a selection of the ones I could attend, and which appealed to me:

M. Engeli (Zürich): A language for 3D graphics applications (with excellent life demo)

H. Klamet (Philips, Hamburg): CAD for the layout of integrated circuits, *CADLIC*

J. Nievergelt (Urbana, US): The automation of introductory CS courses.

J. McCarthy (London): Automatic file compression.

V. Strassen (Zürich): Evaluation of rational functions.

P. Deussen (Karlsruhe): Description of processes.

D. Grüne (Amsterdam): *ALEPH*, a language encouraging program hierarchy.

R. Schild (Zürich): Interactive structured programming.

P. Danielson (Linköping): Micro programming, a hardware point of view.

D. Edwards (Manchester): Hardware innovation and computer design.

All papers are published in: A.Günther, B.Levrat, H.Lipps (eds.):

*International Computing Symposium 1973*, North-Holland/Elsevier, Amsterdam 1974.

Furthermore, I appreciate the interesting discussion with professor Deussen (Karlsruhe) regarding the relation between semigroups and automata, and the possibility of an algorithmic and quantitative approach to structure. To gain historical insight in mathematics and in computers, the following books are of great interest:

E.T. Bell: "The Development of Mathematics", McGraw-Hill, 1945.

P. and E. Morrison: "Charles Babbage and his Calculating Engines", Dover Publ. 1961.

G. Polya: "Mathematics and Plausible Reasoning", (2 vols.) Princeton Univ. Press, 1954.

A comment on the organisation of a symposium of great variety: in such a dynamic interdisciplinary field as computing science, it is very difficult to divide papers among parallel sessions in a manner satisfactory to all. Conflicts of time and place are frequent. It may be worthwhile to consider a much larger parallelism like that of the market place, where each presentator has his own stand with chairs, essential text, diagrams and possibly a demonstration prepared. Such a free format would suit better the purpose of personal dialog and flexible timing. Of course, the collective presentation of invited papers should be used to indicate a direction of research which, to the opinion of the organisers of the symposium, deserves extra attention.

—o000o—