A new science, from birth to maturity

(Talk held at the Anniversary Celebration of the Department of Computing Science, ETH Zürich, on 19 October 1988.)

If we put the birth of Computing Science at the advent of the program-controlled computer, the topic is about four decades old. These have been exciting decades and I am grateful for my good fortune of having been involved for most of that period, grateful because this involvement enabled me to observe that whole process of growth from close quarters. My observations cover the whole gamut, ranging --as they do-- from the stage that Computing Science hardly existed to the current stage in which Computing Science is a vigorous and flourishing discipline whose academic viability is no longer in doubt. An additional reason for personal gratitude is that, during most of my involvement, my travels frequently took me to the other side of the Atlantic Ocean. Such mobility has been essential for observation "from close quarters" because American and European Computing Science evolved quite differently. We shall return to that difference extensively.

Of course I realize that reducing the globe to just two continents is a rather strong simplification of geography, and that, in a comprehensive study of the history of computing science, much finer distinctions would be needed. But I am not a professional historian and hope that you will allow me to make this geographical simplification. In the same vein I hope that you forgive me when, for the sake of simplicity, I cut the forty-year period up into four decades, each of them with a rough characterization: we could characterize the fifties as the decade of hardware, the sixties as the decade of syntax, the seventies as the decade of semantics, and the eighties --tentatively, for the decade is still young-- as the decade of synthesis.

The fifties

My characterization of the fifties as the decade of hardware is not surprising and easily justified. The whole field began with the conception of the program-controlled computer for which von Neumann got the credit that may be due to Turing . Building such a machine so that it was sufficiently reliable was, at the time, at the limits of technology --if not beyond!-- and it is only too understandable that, at the beginning, the physical equipment attracted all the attention.

Accordingly, most of the inventions of the fifties were aimed at higher speeds, larger stores, and greater reliability. I mention as examples: transistors and ferrite cores, the parity check, the B-line --later better known as the index register-- , the real-time interrupt, automatically managed multilevel store, and the timid beginnings of multiprocessing. Those were the days when the goals could easily be quantified: speeds in number of operations per second, store size in bits, and reliability in MTBF. (The last term, standing for "Mean Time between Failures", strongly suggests that machines continued to be expected to fail frequently.) The significance of this decade is best appreciated by realizing how many of these innovations are still with us, in one form or the other.

In all three technical respects, progress was impressive indeed. To give you one example: I remember that we accepted, almost as a Law of Nature, that machine speeds doubled each year, and all of you know that that amounts to a factor of one thousand over the decade. But there is no such thing as a free lunch, and the price we paid was heavy: programming remained a haphazard activity. Firstly, programming was hardly seen as a problem; most people felt that programming required no more than accuracy and enough computer access and sufficiently short turn-around time to debug your programs. Secondly, in those equipment-oriented days, programming was viewed as intrinsically tied to a specific machine; consequently, the greatest expert in the cunning exploitation of specific machine features was regarded as the best programmer. To put it bluntly, the distinction between a programmer and a hacker had not yet been made.

For the sake of completeness: it was not a decade of hardware progress only. The fifties saw a number of efforts to ease the coding problem, such as autocoders and FORTRAN, but, true to the spirit of the decade, each of these coding aids was aimed at a very specific machine. Consequently, it remained very hard to see programming as something to which science could contribute. Finally, the fifties gave us the professional societies such as the ACM and the BCS; true to aforementioned spirit of the decade, their names refer explicitly to the equipment. They were <u>not</u> scientific societies, but professional ones, in which practitioners could exchange their experience.

The sixties

Let us turn our attention to the next decade, the sixties. Its first month witnessed an amazing landmark in the international history of computing, viz. the publication of the ALGOL 60 Report, edited by Peter Naur. Three facts stand out: firstly, ALGOL 60 was truly the outcome of international cooperation in the best sense of the word, secondly, it has never been doubted that it was an achievement of the highest scientific standard, and thirdly, at either side of the Atlantic Ocean, its fate was totally different. At both sides it was instrumental in giving shape to a young science, but the shapes were very different.

ALGOL 60 was a breakthrough by virtue of

0. its block structure
1. its parameter mechanism
2. its inclusion of recursion
3. its inclusion of the type Boolean
4. its "cleanness" (e.g. the absence of special constraints on index expressions)
5. the formal definition of its syntax
6. the nonoperational definition of its semantics (to the extent that it was not immediately clear how to implement it)
7. its stability (due to very few trouble spots)
8. its machine-independence.

Compared to the culture of the day, this is a truly impressive list! Allow me to highlight three seemingly minor points. With its unambiguous identity and scope of local variables, the block structure incorporated a radical improvement over standard mathematical practice in which both identity and scope of dummies are often not defined beyond "but every mathematician knows what is meant". As to recursion, the mathematical definition of a continued fraction was "a fraction whose numerator is an integer and whose denominator is an integer plus a fraction whose numerator is an integer and whose denominator is an integer plus a fraction and so on". The proper definition of a continued fraction is of course "a fraction whose numerator is an integer and whose denominator is an integer plus a continued fraction". [For heaven's sake, don't underestimate the quantum leap of this improvement. A few years ago, I quoted these two definitions of a continued fraction

as an example of progress. I later heard that from this example a famous scientist in my audience, Fellow of the Royal Society and all that, had concluded that I was mathematically incompetent because everyone knew that circular definitions did not make sense.] The introduction of the type Boolean as a first class citizen is a similar milestone: to this very day, the standard mathematician cannot read a Boolean expression without interpreting it as a statement of fact, and for him $0 = 1$ is "wrong", rather than one of the many expressions that has the value false .

ALGOL 60 was conceived at a time that business administration and numerical computations were viewed as the main computer applications, but the mere presence of the ALGOL 60 Report made it immediately obvious that there was more to programming than coding for these two types of applications: some people would have to write the ALGOL compilers! Besides being viewed as data processor and as calculator, the computer became more widely viewed as symbol manipulator as well. The formal definition of the syntax was an invitation to approach the compilation f in a less ad-hoc manner. Moreover, the combination of block structure, parameter mechanism, and recursion offered the possibility of a clean presentation of algorithms of a novel degree of sophistication. The topic was no longer "trivial" and computing science became conceivable as an academically respectable discipline. ALGOL 60 has been more than a scientific landmark: politically, it has been an indispensable stepping stone for Computing Science on its way to academic respectability.

f: problem

As said, the two sides of the Atlantic Ocean received ALGOL 60 very differently. The superficial story is that ALGOL 60 was widely received in Europe but rejected in America, where it failed to dislodge FORTRAN . IBM had penetrated the American system of higher education by giving the more prestigious universities its machines almost for free, thereby hooking those universities, while European academia had had the good fortune of having been spared that largesse and had remained free, etc. Though not without a germ of truth, this story is really too superficial, for even American universities can unhook themselves if they really wish to do so. The story of ALGOL 60's different fates needs refinement, and so does its explanation.

In America, where, within a few years, Perlis coined the term "ALGOL-like

languages", the study of formal languages quickly emerged as a scientific topic in its own right and ALGOL 60's main role was to provide the initial paradigms for that scientific inquiry. It was hardly viewed as a vehicle for serious programming. And it is precisely in that last capacity that ALGOL 60 drew most of the European attention. Europe quickly implemented ALGOL 60 in its full glory and used it; Tony Hoare's "quicksort" convinced the last doubters that the inclusion of recursion was more than a whim, "Jensen's Device" showed the full power of the parameter mechanism, and, thanks to ALGOL 60 , programming could, and did, become serious business.

The next decade would drive that difference in attitude home to me. At conferences in the USA, the following conversation became a standard ingredient.
Question: "And, Dr.Dijkstra, what are you currently working on?"
Answer: "Programming."
Reaction: "Ah, I see: Programming Languages. How interesting."

But I am rushing ahead. Both camps took ALGOL 60 seriously, both viewed it as an opportunity and incentive for scientific investment, but they stressed different aspects, viz. formal languages versus programming. How come?

There was an obvious difference in timing. World War II had left the USA with a thriving economy and industry, while those were a shambles in Europe. By 1960, Europe's recovery was well under way, but powerful computers were still much less common than in America. It is understandable that the need for something that would deserve the name of Computing Science was felt more urgently in America than in Europe. Moreover, the fence around the campus that separates the academic world from the rest of society is traditionally much lower than in Europe. Finally, the legal status of most European universities is such that establishing a new academic discipline usually requires a few decades of lawmaking. Finding out how the techniques of scientific thought could be applied to meet the programming challenge was a less tangible and more distant goal than developing formal language theory: the latter topic was much more in line with the mathematical tradition of the day and thus a target within reach. Urgency and opportunity thus made it in the sixties a cornerstone of America's budding computing science. So much for the difference in timing.

Another explanation for the different reactions to ALGOL 60 can be found in a traditional difference in technological priorities. Right from the start, American computing has been much more concerned with attaining speed than with reducing equipment, be it circuitry or storage size. There is a very simple economic/technological explanation for this: after World War II, none of the European laboratories had the resources needed for the development of the fastest machines conceivable at the time. But that is only part of the explanation, for there is also a cultural difference, as mentioned in passing by Alice S.Rossi* in 1964: "Americans are easily impressed by large numbers.". By the time ALGOL 60 came around, this aspect had already created two completely different computing cultures. I remember a conversation in 1962, in Rome. We were sitting around a coffee table. One American boasted that he had made an "algebraic translator" of 50,000 instructions, only to be immediately outdone by one of his compatriots, whose algebraic translator comprised no less than 80,000 instructions. Peter Naur broke the subsequent silence of awe by remarking that he had written an ALGOL translator of 5,500 instructions, upon which I could outdo him with a compiler of only 2,700 instructions. In short: our yardsticks for achievement measured in opposite directions!

The American priority to speed had two direct consequences. Firstly, American computing all but ignored, as too time-consuming, the closed subroutine, which, right from the start, was the cornerstone of European computing. This attitude would culminate in the design of the IBM/360 whose large number of explicitly named registers actually defied a reasonable implementation of closed subroutines. Secondly, American computing ignored ALGOL 60 as a serious programming vehicle because --rightly or wrongly-- it was felt that ALGOL implementations were too slow.

The sixties were the decade in which automatic computing began to gain academic respectability, and academic departments started to emerge. The American departments were, on the whole, sooner to be institutionalized. They were also more ready to incorporate application areas as part of the departmental responsibility, and to this very day you can find in American universities areas such as numerical analysis and artificial intelligence being a part of the CS Depart-

* In fact in a footnote to her article "Equality between the Sexes: An Immodest Proposal".

ment.

Further American contributions from the sixties were symbolic processing --I mention LISP-- and complexity theory. They were gratefully incorporated in the American CS curriculum. Were the American departments premature? Many people felt that way and the question whether Computing Science deserved the name of a science was raised over and over again. Newell, Perlis, and Simon have tried to settle that discussion at the end of the decade by "Facts breed science. Computers exist. Ergo.". Needless to say, this did not end the discussion.

European universities too started to think about computing science. They, too, thought deep and hard about forging an academic discipline worthy of the name. But they were less in a hurry to institutionalize and eventually they embarked with a probably somewhat more conscious design. For an academic discipline to be viable, its areas should be coherent and should mutually reinforce each other; moreover, the material taught should have a staying power of, say, fifty years. For the sake of coherence, and in recognition of the fact that the automatic computer really deserves the name "general purpose", it was generally decided that the application areas had better not be included in computing science. For the sake of staying power, it was generally decided that computing science should dissociate itself from the fickle market place: teaching how to make do with the equipment currently on the market was not the calling of European computing science, and all material with a half-life of five years was banned. In passing I mention that COBOL and FORTRAN were viewed as industrial products, and therefore not taught.

Europe's contribution to computing in the sixties consisted mainly in improving the art of design, such as design of operating systems, of programming languages and of their implementations. Quite a few of such designs were engineered with a novel delicacy. Furthermore, thinking removed itself from the physical equipment, and programming languages were no longer understood in terms of their implementation. (When Perlis called ALGOL 60 "a very inefficient language", the Europeans in his audience were shocked by his patent lack of separation of concerns.)

To complete the picture of the sixties, in 1968, the existence of the software crisis was admitted at the NATO Conference on Software Engineering in Garmisch-

Partenkirchen, and Wirth started the implementation of PASCAL . In 1969, C.A.R.Hoar published his "An axiomatic approach to computer programming" and I circulated my "Notes on structured programming". Let us move on to the next decade.

The seventies

I called the seventies the decade of semantics: the interest in syntax and parsing tapered off, Dana Scott's denotational semantics provided the logical foundation for recursion in all its glory, and formal proofs of program correctness entered the picture.

In hindsight, the reduction in interest in syntax and parsing was more than justified. Programs that, on account of complexity of the syntax, are hard to parse mechanically are correspondingly hard to compose correctly; consequently, a straightforward syntax that simplifies the parsing eases the programming task (something the designers of Ada overlooked).

As in the mean time can be expected, the American and European interests in proofs of program correctness differed greatly: the American interest focussed on the mechanization of a posteriori verification of given programs, written in given languages; the European interest was more in a constructive approach to the problem of program correctness, and turned to design methodologies or calculi for the derivation of programs that would be correct by construction. From the European perspective, a posteriori program verification amounted to putting the cart before the horse; from the American perspective, the constructive approach of the Europeans was hopelessly idealistic, because it was mathematical. This was the decade in which I began to stress that as scientists we should clearly distinguish between the intrinsic problems of automatic computing and the problems caused by shortcomings of the American educational system, which traditionally does not consider intellectual advancement its primary concern.

On the American academic scene, complexity theory continued to flourish; automatic theorem proving attracted a lot of attention and got off the ground. A serious concern during the seventies was how to prevent a sizeable part of Computing Science from deteriorating into the dishonourable art of "How to live with the IBM/360"; in winning this battle, the less infected and intellectually

more autonomous universities in Europe played a considerable role. Finally, the advent of the personal computer revived most of the mistakes of the fifties --but now on a more grandiose scale-- . Today, the ubiquitous personal computer has created an equally ubiquitous misunderstanding of what computing science is about. (At a party, two years ago, Tony Hoare and I were approached by a mathematician who expressed his delight in meeting two such outstanding computing scientists for he had never been able to understand what he had to do in order to save a file on his personal computer, model so-and-so. We could not help him and Tony asked him, whether he could recommend a book on category theory; he could not help Tony either.)

The seventies were a decade of frantic and expanding activity; at the same time it revealed symptoms of decay and showed that in the academic community some intellectual rot was setting in. Also in this respect, America was leading: it became highly productive in rather uninspiring papers, ranging from boring to utterly foolish. Soon, Europe would follow this example.

In the seventies, universities all over the Western world had a hard time. Firstly, lecturing style deteriorated when chalk and blackboard were ousted in favour of the overhead projector with prepared foils: instead of lectures, students got presentations. Secondly, life on campus had been totally disrupted by the student revolts of the late sixties, which breathed an anti-intellectualism as vigorous as Chairman Mao's Cultural Revolution of which they were the echo. All disciplines have suffered from these two calamities; we must bear in mind that they hit Computing Science at a very vulnerable stage, and there are reasons to suspect that American computing suffered even more than its European counterpart.

The American universities are generally praised for the fact that they are much less a world apart and that the barriers to communication across the campus boundary are much lower than in Europe. It has its charms, but the price can be heavy. For a university to be leading, it must offer what society needs, rather than what society demands. The lower barriers make that the direct demands of society are heard much more loudly and it is not surprising that American computing science, on the whole, became more led than leading.

I am not referring to the local banker's pressure for more "Advanced COBOL" in the curriculum: all but the very smallest institutions can resist such pressure. I am referring to the conflicting pressures from a society that is traditionally ambivalent about technology, that welcomes gadgets almost without restriction but equates technological expertise with the loss of innocence that is an essential ingredient of The American Dream. In the case of Computing Science I am referring to the pressures from a society that is structurally unable to include in its vision a view of programming as a branch of formal mathematics and applied logic, and that therefore is forced to ask for snakeoil. If, then, the campus is insufficiently shielded from those pressures and the market forces are given too free a reign, you can draw your conclusion: a flourishing snakeoil business, be it in "programming by example", "object-oriented programming", "natural language programming", "automatic program generation", "expert systems", "specification animation", or "computer-supported co-operative work". (I am not inventing these slogans: they are all honest quotations.) I used the term "decay" and am forced to conclude that I did not exaggerate.

A more universal problem is presented by the negative effects of academic institutionalization: in the seventies, they became quite visible for computing science. Firstly, the ridiculous fragmentation: these days we see special professors in "local-area networks", "compiler construction", or "support environments". Secondly, the diversion into elaborate trivia, caused by the Ph.D. mechanism that requires a steady stream of well-delineated and obviously solvable problems (if you care for them). Thirdly, the need for an outlet for your less than gifted students (which has introduced such topics as "human factors", "software metrics" and "experimental computing science"). As computing scientists, we don't need to feel particularly guilty: no academic discipline has found a truly satisfactory solution for the education of the unavoidable second- and third-rate researchers. Hence, we usually ignore the dilemma, or deny its existence, but it is a negative effect of academic institutionalization.

The eighties

Let us now switch to the current decade. I shall be less elaborate on that one for two obvious reasons: firstly, it is not over yet, and, secondly, it is still too close to have established its significance.

European computing science suffered from more than the traditional problems caused by institutionalization, because many departments were founded in such a hurry that quite a few vacancies were filled by so-called "instant professors". Moreover, many a government forced its universities into co-operative efforts with industry (a trend from which Computing Science suffered more than, say, Assyrian Linguistics). As a result, even genuine research potential has been diverted to the ephemeral, shallow, or foolish. Confining myself to Computing Science proper, I shall ignore these aberations.

Tentatively, I called the eighties the decade of synthesis. I did so because Computing Science and Formal Mathematics got more and more intertwined, and did so in a variety of ways.

Firstly, mechanical theorem proving or proof verification came of age. It has definitely left the youthful stage of toy problems: successful applications have ranged from deep theorems such as Gödel's Theorem, to practical challenges such as proving the correctness of circuitry and special-purpose operating systems. The appearance of a journal dedicated to Automated Reasoning is a clear signal.

Secondly, the fundamentally close connection between proving and programming, as revealed by Constructive Type Theory, is beginning to have its impact: programs are deduced from constructive existence proofs and vice versa.

Thirdly, logic programming has established a visible link between proving and computing, a link that also emerges in the

application of complexity theory to the notion of provability. The discovery of the existence of correct theorems that shall never be proved because the price of the proof is prohibitive --and that is putting it mildly-- caused some rethinking about the scope and purpose of mathematical reasoning.

Fourthly, it is beginning to be realized, mainly on campus but even in some pockets in industry, that the programming task presents a fertile field for the application of the techniques of scientific thought. The emergence of the journals "Science of Computer Programming" at this decade's beginning and "Structured Programming" at its end are clear symptoms. It is fascinating to observe how this development works both ways and how, in the wake of Programming Methodology, the somewhat wider topic of Mathematical Methodology is emerging. [Since this was written in first draft, Springer announced yet another journal: "Formal Aspects of Computing", with the telling subtitle "The International Journal of Formal Methods".]

It is fascinating to observe how a breakthrough in our manipulative abilities has created the opportunity of realizing an increasing portion of Leibniz's Dream of presenting calculation, i.e. the manipulation of uninterpreted formulae, as an alternative to traditional mathematical reasoning. Because the Dream of Leibniz aims at providing an alternative for traditional mathematical reasoning, traditional mathematics did not provide the most hospitable environment for its realization; this, therefore, had to take place in a separate discipline, which is now known as Computing Science.

It is not surprising that the computing scientist picked up this gauntlett. By virtue of its mechanical interpretability, each programming language presents a formal systems of some sort, and so the notion of a formal system is something he grew up with. He is very familiar with formal methods because they provide the

only sufficiently reliable way of designing programs. Knowing, for instance, how compilers work, he is familiar and totally at ease with the idea of manipulating uninterpreted formulae. Finally, he has good reason for not sharing the common fear of symbol manipulation because he has the tools for mechanization at his disposal.

It is my conjecture that meeting the challenge, embodied in the Dream of Leibniz, will provide the ultimate justification for the existence of Computing Science as a scientific discipline in its own right. We have to show that it is possible to be precise and complete without making a mess of it.

Let me quote Leibniz to wish us well for the next forty years and beyond: "Calculemus!".

I thank you for your attention.

Austin, 2 October 1988

prof.dr.Edsger W.Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 - 1188
United States of America

Post-delivery comment My Zürich audience was generally of the opinion that I had been too kind to the Europeans, and I agree. For instance, I forgot to mention all the interdisciplinary nonsense done in the wake of the official French definition of "l'Informatique". European computing science has been disgracefully slow in dissociating itself, for instance, from business administration; for this, the absence of Business Schools was no excuse. (End of Post-delivery comment.)