The threats to computing science

Let me describe to you by way of introduction the observation that induced me to choose "The threats to computing science" as my topic for this keynote address. The observation is simply that, when browsing through a computing science journal or conference proceedings, one usually sees at a first glance the geographical origin of the work described.

Now, in any area we have always had national differences, but their obviousness in computing becomes surprising as soon as we remember all the homogenizing forces at work during the few decades in which the topic emerged, and allow me to mention a few of them, just to underline the strength of this argument.

Firstly, right from the start, English has been the Lingua Franca of computing: as early as 1956, when as a young Dutchman I gave my first lecture abroad, I did so —though at a German computing conference— in English. And as soon as, at either side of the Atlantic Ocean, the ACM and the BCS started their journals, these journals became the leading ones in the field.

Secondly, at an early stage, the computing scientists of the world became a travelling lot: the year of the UNESCO Conference in Paris,

1959, is about the time when the jet airliner became more common.

Thirdly – and this process had already started in the 50's – the computing community began to use more and more the same few machines and the same few programming languages. Even behind the Iron Curtain they could think of nothing better than faithfully copying the IBM/360 and are now busily implementing Ada at the request of the Red Army.

And finally, to deprive you of the last visual clues, people are beginning to use the same laser printers and the same text editing software. Gone are the days when Niklaus Wirth unmistakably identified himself by his typewriter and you could recognize his writings even when they were held upside down.

In short, in the young discipline of computing science, the homogenizing forces have been stronger than in any other discipline I can think of. And yet, the geographical origin of most of its publications is patently obvious.

* * *

I can think of only one explanation of this curious phenomenon, and that is the combination of strong local pressure and malleable computing groups that yielded only too easily to those

pressures. Let us deal with the malleability first and with the pressures later.

The malleability is understandable when we realize that at the beginning the computing community was very uncertain as to what its topic was really about and got in this respect very little guidance from the confused and confusing world by which it was surrounded.

The Fathers of the field had been pretty confusing: John von Neumann speculated about computers and the human brain in analogies sufficiently wild to be worthy of a medieval thinker and Allan M. Turing thought about criteria to settle the question of whether Machines Can Think, a question of which we now know that it is about as relevant as the question of whether Submarines Can Swim.

A further confusion came from the circumstance that numerical mathematics was at the time about the only scientific discipline more or less ready to use the new equipment. As a result, in their capacity as number crunchers, computers were primarily viewed as tools for the numerical mathematician, and we needed a man with the vision of Stanley Gill to enunciate that numerical analysis was for the computing scientist like toilet paper for the sanitary engineer: indispensable when he needs it.

But the greatest confusion came from the circum-

stance that, at the time, electronic engineering was not really up to the challenge of constructing the machinery with an acceptable degree of reliability and that, consequently, the hardware became the focus of concern. This distortion is faithfully reflected in the names coined in those days: ACM stands for Association for Computing Machinery, BCS stands for British Computer Society, and Universities founded in those days Departments of Computer Science. (The British Universities, a little bit slower to react, were by then a little bit wiser and erected Departments of Computing Science.)

I called this focus on hardware a distortion because we know by now that electronic engineering can contribute no more than the machinery, and that the general purpose computer is no more than a handy device for implementing any thinkable mechanism without changing a single wire. That being so, the key question is what mechanisms we can think of without getting lost in the complexities of our own making. Not getting lost in the complexities of our own making and preferably reaching that goal by learning how to avoid the introduction of those complexities in the first place, <u>that</u> is the key challenge computing science has to meet.

Nowadays machines are so fast and stores are so huge that in a very true sense the computations we can evoke defy our imagination. Machine capa-

cities now give us room galore for making a mess of it. Opportunities unlimited for fouling things up! Developing the austere intellectual discipline of keeping things sufficiently simple is in this environment a formidable challenge, both technically and educationally.

As computing scientists we should not be frightened by it; on the contrary, it is always nice to know what you have to do, in particular when that task is as clear and inspiring as ours. We know perfectly well what we have to do, but the burning question is, whether the world we are part of will allow us to do it. The answer is not evident at all. The odds against computing science might very well turn out to be overwhelming.

Since the Romans have taught us "Simplex Veri Sigillum" — that is: simplicity is the hallmark of truth — we should know better, but complexity continues to have a morbid attraction. When you give for an academic audience a lecture that is crystal clear from alpha to omega, your audience feels cheated and leaves the lecture hall commenting to each other: "That was rather trivial, wasn't it?" The sore truth is that complexity sells better. (It is not only the computer industry that has discovered that.) And it is even more diabolical in that we even use the complexity of our own constructs to impress ourselves. I have often been impressed by the cleverness of

my own first solutions; invariably the joy of the subsequent discovery how to streamline the argument was tempered by a feeling of regret that my cleverness was unnecessary after all. It is a genuine sacrifice to part from one's ingenuities, no matter how contorted. Also, many a programmer derives a major part of his professional excitement from not quite understanding what he is doing, from the daring risks he takes and from the struggle to find the bugs he should not have introduced in the first place.

These were internal complications. For a better understanding of the external pressures we had better view our topic for a moment in the wider context of science and society in general. Because computers appeared in a decade when faith in the progress and wholesomeness of science and technology was virtually unlimited, it might be wise to recall that, in view of its original objectives, mankind's scientific endeavours over, say, the last five centuries have been a spectacular failure.

As you all remember, the first and foremost objective was the development of the Elixir that would give the one that drank it Eternal Youth. But since there is not much point in eternal poverty, the world of science quickly embarked on its second project, viz. the Philosopher's Stone that would enable you to make as much Gold as you needed.

Needless to say, the planning of these two grandiose research projects went far beyond the predictive powers of the seers of the day and, for sound reasons of management, the accurate prediction of the future became the third hot scientific issue.

Well, we all know what happened as the centuries went by: medicine, chemistry, and astronomy quietly divorced themselves from quackery, alchemy and astrology. New goals were set and the original objectives were kindly forgotten.

Were they? No, not really. Evidently, the academic community continues to suffer from a lingering sense of guilt that the original objectives have not been met, for as soon as a new promising branch of science and technology sprouts, all the unfulfilled hopes and expectations are transferred to it. Such is the well-established tradition and, as we are all well aware, now computing science finds itself saddled with the thankless task of curing all the ills of the world and more, and the nett result is that we have to operate in an unjustified euphoria of tacit assumptions, the doubting of which is viewed as sacrilege precisely because the justification of the euphoria is so shaky. Let me cast a few doubts, be it at our own peril.

The fundamental dogma of the euphoria is, of course, that whatever we can do, machines can do it better and cheaper, and, as a corollary, that the use

of machines leads to a better product and saves time and money. The dogma continues to be accepted notwithstanding the fact that we observe almost daily that in many respects the product has deteriorated.

In June, during my last preparatory visit to the USA, I tried to order a telephone to be installed upon my arrival in September. The lady reacted as if I had made her an indecent proposal: didn't I know that after 30 days I "would be dropped out of the computer"? This was not incidental, this was characteristic; just think how often the bank has denied you a reasonable service on the ground that "the computer won't let them do it".

These are big installations, but the small ones aren't any better. As a referee I have to judge many manuscripts and the ones prepared on word-processors are invariably the worst, qua printing quality, or qua layout, or qua style, notation, and contents. The proposed style of composing —write first, improve later— rarely leads to a text from which all ill-considered turns have been weeded out. Finally, the suggestion that the proposed style of composing iteratively would save time is an obvious and blatant lie. And yet the equipment is sold by the millions...

Does this overestimation of the usefulness of

the gadget hurt computing science? I fear it does. At the one end of the spectrum it discourages the computing scientist from conducting all sorts of notational experiments because "his word-processor won't allow them", at the other end of the spectrum the art-and-science of program design has been overshadowed by the problems of mechanizing program verification. The design of new formalisms, more effective because better geared to our manipulative needs, is neglected because the clumsiness of the current ones is the major motivation for the mechanization of their use. It is not only the performing artist who is, in a very real sense, shaped by the instrument he plays; this holds as well for the Reasoning Man, and I leave it to you to determine how disturbed you are going to be by this observation.

Back to the history of our subject. During the early days the concern to get the hardware more or less in working condition overshadowed all others and we naively thought that, once the hardware was reliable, our problems would be over, little realizing that our problems in using the equipment would really start only then. The confrontation with the difficulty of programming almost caused a severe dent in the faith in how wonderful computers were going to be. The solution to this moral dilemma was as simple as effective: the intrinsic difficulty of programming was just denied and,

if problems had to be admitted, the so-called "primitive nature" of the machines was blamed and it was promised that with "the next generation" the problems would no longer occur.

Now here is a very real threat to computing science. The intrinsic difficulty of the programming task has never been refuted, it has only been denied because admitting it was socially unacceptable. Not surprisingly, research in programming methodology has not flourished in societies unable to admit that the programming problem was serious.

I vividly remember from the late 60's the tendency to blame the programming languages in use and to believe in all naively that, once the proper way of communicating with the machines had been found, all programming ills would have been cured. To give you only one example, the ACM Conference on the Principles of Operating System Design in Gatlinburg, Tennessee, 1967, needed only one day to come to the rash conclusion that we did not know how to design operating systems because we did not have a language to design them in! If we only had the right programming language! The fact that operating system design posed some at the time tough conceptual and logical problems was hardly allowed to surface. Consequently, those problems did not get the attention they deserved, and this had far-reaching consequences: I recently read a series of articles on the Shuttle on-board software and, though President Reagan

has generously offered a teacher a free ride, I tell you I am not available.

The quest for the ideal programming language and the ideal man-machine interface that would make the software crisis melt like snow in the sun had -and still has!- all the characteristics of the search for the Elixir and the Stone. This search receives strong support from two sides, firstly from the fact that the working of miracles is the very least you can expect from computers, and secondly from the financial and political backing from a society that had always asked for the Elixir and the Stone in the first place.

Two major streams can be distinguished, the quest for the Stone and the quest for the Elixir.

The quest for the Stone is based on the assumption that our "programming tools" are too weak. One example is the belief that current programming languages lack the "features" we need. PL/I was one of the more spectacular would-be stones produced. I still remember the advertisement in Datamation, 1968, in which a smiling Susie Mayer announces in full colour that she has solved all her programming problems by switching to PL/I. It was only too foreseeable that, a few years later, poor Susie Mayer would smile no longer. Needless to say, the quest went on and in due time a next would-be stone was produced in the form

of Ada (behind the Iron Curtain perceptively referred to as PL/II). Even the most elementary astrology for beginners suffices to predict that Ada will not be the last stone of this type.

Ada will not meet its major objective, viz. that of reducing software costs by standardization, and it will not be the vehicle for programs we can rely upon, for it is so complicated that it defies the unambiguous definition that is essential for these purposes. Long before the design was frozen, computing scientists from all over the world have given plenty of warning but the political body in question preferred to ignore these warnings and to decide on a design that cannot be salvaged. From a scientific point of view all further attention paid to Ada is a waste of effort. But the sheer buying-power of the DoD makes Ada an undeniable reality, which in combination with DARPA's policies for the funding of software research can only increase the pressure to spend research effort on the wrong problems.

Another series of stones in the form of "programming tools" is produced under the banner of "software engineering", which, as time went by, has sought to replace intellectual discipline by management discipline to the extent that it has now accepted as its charter "How to program if you cannot."

In parallel we have the search for the Elixir.

Here the programming problem is simply solved by doing away with the programmer. Wouldn't it be nice, for instance, to have programs in almost plain English, so that ordinary people could write and read them? People tend to forget that "doing away with the programmer" was COBOL's major original objective. Fifteen years later, 80 % of the world's programming force was absorbed by the use of COBOL, a figure which gives us some idea of how effective that elixir has been. Elixirs are typically implemented – to quote Grace Hopper's own words – by "education of the computer".

Since then we have had elixirs in a countless variety of tastes and colours. I have fond memories of a project of the early 70's that postulated that we did not need programs at all! All we needed was "intelligence amplification". If they have been able to design something that could "amplify" at all, they have probably discovered it would amplify stupidity as well; in any case I have not heard from it since.

The major attraction of the modern elixirs is that they relieve their consumers from the obligation of being precise by presenting an interface too fuzzy to be precise in: by suppressing the symptoms of impotence they create an illusion of power.

Finally I must mention the now fashionable inter-disciplinary research aimed at making Stone-and-Elixir all in one. I refer of course to the user-friendly programmer's workstation, the interactive learning tool, the integrated project support environment, and the fully automated high-resolution creative dream manufacturing management system.

* * *

So much for the unrealistic expectations from any novel branch of science or technology. In the case of automatic computing, the situation is further aggravated by two circumstances.

Firstly, the public at large has very little feeling for the conceptual challenge implied by any non-trivial computer usage and tends to confuse - if I may use an analogy - the composing of a symphony with the writing of its score. As a result its expectations remain unchecked by an understanding of the limitations.

Secondly, I am sorry to say, both the computer industry and the educational business contributed more than their fair share of misleading the public. I think of the company advertizing "Thought Processors" or the college pretending that learning BASIC suffices or at least helps, whereas the teaching of BASIC should be rated as a criminal offence: it mutilates the mind beyond recovery.

And all this wide-spread misappreciation of what automatic computing is all about is a threat to computing science because it tends to divert the research effort into directions in which science can not -and hence should not try to- contribute. At the same time it explains the geographic differences I alluded to in the beginning of this talk: each society has its own specific problems and worries and hence asks for its specific Elixirs and Stones.

Take, for instance, "user-friendliness". Taken literally, this is like the term "motherhood": nobody can be against it, so it means nothing. And hence, if the term "user-friendliness" is given a meaning, it must be a terrible euphemism for something else. The catalogue of the average textbook publisher reveals the secret: the textbook recommendation that is deemed to be most effective is that the book is almost totally unmathematical. Mathematics, with its potential for and hence its requirement of rigour and precision, is evidently the pinnacle of user-unfriendliness. Conversely, a paper full of user-friendly topics is primarily respectable in a- or even anti-mathematical circles. (Personally I think the world could benefit from an International League for the Derision of User-Friendliness.)

There are, however, encouraging symptoms that the period in which each newly coined slogan

could overnight be turned into a respectable research topic is drawing to a close, and those symptoms go beyond the button I received last year with the text "Stop BASIC before it stops you."; the fact that the Siberian Branch of the USSR Academy has launched a serious effort to prevent BASIC from being introduced at Soviet high schools is a more telling symptom.

From this country the bad news is that at one of its great Universities a specially created Vice President of Educational Computing has decided that all their undergraduates should have enough computing power at their disposal but that this required only equipment and no further education "since our kids already know how to program when they leave high school". The good news, however, is that this Vice President made himself the laughing stock of the company -with the possible exception of the company of his co-physicists-.

And also for the American Universities the tide may be turning. Traditionally they have been asked to train the work-force for the American industry, while the question of educating the industry so as to be worthy of their graduates was left untouched. But, currently, companies in Silicon Valley seem to be folding up at a higher rate than they are erected. I consider that good news because it could drive home the message that neither slogans like "knowledge-based decision aids", nor a combination

of adhoccery and brute force will do the job. (It is regrettable that large groups only come to their senses after their day-dreams have turned into nightmares, but, this being so, we should occasionally welcome the nightmares.) During the last decades the American Departments of Computing Science have severely suffered from a discrepancy between what society asked for and what society needed, but, be it slowly, the gap seems to be closing.

As I said earlier, the programmable computer is no more and no less than a handy device for the implementation of any thinkable mechanism. As such it poses on us the burden to demonstrate which mechanisms we can think of sufficiently clearly. It implies the challenge of blending Engineering with the techniques of Scientific Thought; this challenge is exciting and we are ready for it.

(Delivered at the ACM 1984 South Central Regional Conference, November 16-18, Austin, Texas.)

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