transcription

Science fiction and science reality in computing

There is a lot of misunderstanding about computing science, among both computing's practitioners and the public at large, and the purpose of this talk is to clear up that misunderstanding, for it hurts all of us in more than one way.

On the one hand, the achievements of computing science are sadly ignored. Many projects in which the conscious application of the techniques and methods of computing science is indispensable are carried out in a totally unscientific manner as if computing science did not exist. This is a great pity, for it leads to many a costly failure that could have been avoided, and computing science is denied the due recognition for its achievements.

On the other hand, the expectations of what computing science will achieve for us are often totally unrealistic: the performance of miracles by the dozen seems to be the absolute minimum. And this is a great pity too, for it leads to false hopes and unrealistic planning, and when those wonderful plans fall through, computing science will be written off as just another form of quackery.

If you so desire you may view this talk as an effort to change what seems now to be the computing scientist's predicament: first he is not allowed to have any influence at all, and afterwards he will be blamed for all that went wrong.

At first sight one might think that explaining the quintessence of a science and then drawing conclusions as to its appropriate rôle is a straightforward technical job, but regrettably it isn't. The problem is that the message runs counter to too many vested interests and is for many too uncomfortable to be heard. In the past it could be delivered. It used to cause a momentary discomfort, and then the audience would laugh it off and return to the order of the day, tolerating the messenger very much as the court jester is tolerated. But things are changing, the vested interests have increased and been diversified, and more and more people are beginning to realize that this is no longer a laughing matter. While a knowledgeable and dispassionate discussion of these matters has become more and more urgent, it has become more and more difficult to raise the topic. For further technical details I refer to the fate of Galileo Galilei.

* * *

Earlier I said that achievements of computing science are sadly ignored. Well, if it were just that, it would not be alarming: there is by necessity a lag between the useful achievement in the laboratory and its general exploitation. In our case, however, it is alarming because over the last decades the gap between computing science and computing practice has only widened. Whereas computing science made great strides towards a vigorous, rigorous discipline, computing practice showed mainly stagnation. I am not exaggerating: the physicists still think that FORTRAN is the last word in computing, the chemists continue with BASIC, and what APL is for the electronic engineer, COBOL is for the MBA. The human tendency to get attached to the sources of one's misery has been hailed as the great stabilizing factor in many marriages and religions, but by their morbid attachment to those inadequate vehicles those disciplines have manoeuvred themselves into a position in which they are beyond help.

These observations used to apply to the "casual user", but in the mean time the problem has been magnified by the fact that it has affected many university curricula that, instead of training first-class scientists, now train third-rate programmers.

Well, you might say, these people don't consider themselves programmers but physicists, chemists etc., but in the professional field the picture is equally bleak. We have all heard of the Shuttle that did not get launched due to an error in the synchronization software, so I shall give you another example. Recently, British Rail has installed its first computerized signalling system along one of its tracks and they advertised the system in the hope of selling it to other railroad companies by revealing that, in order to avoid the risk of using a compiler, the system had for safety's sake been written in machine code. It was evidently taken for granted that commercially produced compilers play dirty tricks on you. Another area rich in horror stories is provided by VLSI design; among its usual "tools" is a program --very expensive to run-- that tries to reconstruct the circuits from the raster, needed because the program producing the raster cannot be trusted. But the alchemy of communication protocols probably beats them all; originally designed by telecommunication engineers to compensate for incidental hardware errors, they have become so unwieldy that they are responsible for the introduction of many more transmission failures than they were intended to correct and that the verification of their designs has become a major challenge: unravelling that mess can now be a Ph.D. topic. And all that inadequacy was avoidable, for computing science initiated its penetrating and successful study of the problems of compilation and synchronization more than twenty years ago. So much for the gap between science and practice; now for the unrealistic expectations.

We have already seen that curricula are being changed on the unchallenged assumption that, in this modern day and age, computer usage is essential to becoming a good scientist. Now this is very strange, for it is almost impossible not to challenge that assumption in view of the large number of very successful scientists we know that never used a computer because in their days there weren't any. It is true that microcomputers have been sold as aggressively as encyclopedias, blackmailing the parents with the future of their kids. And if you read between the lines, the message was even more seductive: give your kid our homecomputer, and he'll become a genius. But we cannot blame the industry for that misconception, no matter how hard it tries to push its goodies down our throats: in those advertisements they just appealed to the ever-present lurking desire for the Philosophers' Stone that makes gold and the Elixir that gives eternal youth and health. Industry was no worse than the common quack.

To the best of my knowledge it was not industry that invented the term "computer literacy", but society. Quite early in the game it was firmly implanted in the people's minds that computers were there to save our economies (and later also the economies of the countries of the third world). For a while, no one knew how, but fortunately someone hit upon the bright idea that information was "a resource", so that solved that problem, the more so since, for the first time in the history of mankind, we had now a valuable resource of infinite supply. Then people remembered how oil had created immense riches for a happy few; this time that should not happen again, so "computer literacy" for the millions was invented to solve that problem. What that term should mean is still an open question, but that did not prevent our visionaries from happily proceeding to invent the post-industrial society in which people, if not happily teleconferencing, would devote their ample leisure time to creative video-games.

There is nothing novel in people expecting that Heaven on Earth will be established in the near future; novel, however, is that microelectronics will do that job, rather than the Grace of God. I am not exaggerating: expectations are so unrealistic that what should be reasoned discourse sounds more often than not like people participating in a strange ritual, faithfully reciting prayers in a language they don't understand. Let me just quote you a few recent examples.

The director of a large research laboratory, when interviewed at the occasion of his retirement, extolled the virtues of the home computer by stressing that it would enable people "to order their creative thoughts". Now think for a while about the nonsense! Between you and me, when did you have your last creative thought? Two years ago? Not bad. It would have made more sense if he had said "to order the stamps in your collection".When Bertrand Russell made the famous remark "Many people would sooner die than think. In fact they do.",he showed more realism. My private conclusion was that that director retired none to soon.

At an international conference on Computers and Education it was immediately agreed that the little kids should be introduced to computers as young as possible. There was a moment's concern that such an intensive mechanization of the educational process might hamper the little kids in their emotional growth and in the development of their contactual skills, until someone remarked that "talking to a program could do no harm because, being a product of the human mind, a program was essentially human". Now that I come to think of it, was't also Hitler's Final Solution a product of the human mind?

At a recent conference on the feasibility or infeasibility of the software for the Star Wars project, one of the arguments in favour of feasibility, I have been told, was the remark that today almost all of the banking operations had been successfully computerized, hadn't they? Well, as a matter of fact, they haven't: as late as 1986 the salary payment of a large (computer!) company was electronically screwed up. But even if they had, it would have been a fake argument: the two projects are so different that the analogy is several orders of magnitude too shallow to be of any value. The worst of all, I think, was that this fake argument was put forward by someone who had found himself a place in the academic world.

* * *

In the sixties Sir Hermann Bondi wrote an article on who should do academic research. The question was an urgent one. Prior to the second world war scientists were a negligibly small minority, but in response to the recognition that that war had been won by science and technology, the academic enterprise was boosted thereafter. Not surprisingly, Bondi came to the conclusion that many of his contempories engaged in scientific research had really better be in some other business. Because the explosive growth of the university was directly related to the expected high benefits from science, Bondi also devoted a section of his paper to the usefulness of science in general. His conclusion is sobering.

He points out that of the problems coming from outside the campus, from "the real world", about 80 % are trivial and about 20 % patently unsolvable, and that academic research has its potential impact in the thin boundary layer in between as it is only there that knowledge, talent and hard work can achieve something that cannot be achieved by other means. In the scientific/technological euphoria of the day, such a sobering warning was more than needed; it was, however, neither heard nor heeded, but that is another story.

By an unfortunate accident of history, computing emerged during just those decades of unbounded faith in the wholesomeness and power of science and technology. Whereas the more established sciences had their roots and traditions in soberer times, we embarked on computing with our expectations unbridled. From a historical perspective, the naive optimism is quite understandable. And there may even be some justification for it. You see, Bondi is a physicist and his 80/20 % judgement was derived from his knowledge of how the natural sciences deal with the complexities that we are faced with in the world that surrounds us. Computing science, though, deals with a world of artefacts in which the complexity is of our own making. That makes a big difference and it is quite possible that, in computing, Bondi's "boundary layer" is not so vanishingly thin at all. But in order to make an educated guess of what that layer could and should comprise, we have to understand how computing science emerged and grew to operate.

Before turning to computing science specifically, however, I must dismiss one further dream that was frequently dreamt in those decades, viz. that it is possible to "plan" research so as to produce in due time what the world asks for. Science does not work that way (which, by the way, is a good thing as there is often a grave discrepancy between what the world asks for and what the world needs). E.T.Bell justly praises a number of rulers for being "far-sighted enough to see that the simplest way of getting mathematics out of a mathematician was to pay his living expenses", and the same holds for the other sciences. No dramatic progress of science ever occurred because some benefactor commissioned the result. Significant progress in science only occurs when after probing pondering a knowledgeable original mind has concluded that something baffling may now be ripe for understanding or something very difficult may now be done. Successful scientific research is the art of doing the just possible, and consequently scientific development is much better regarded as an autonomous process with its own private rules than as a planned activity with external objectives. (And this is the paradox faced by all directors of industrial research laboratories: after having attracted the right people, they cannot serve their company better than by leaving those people alone.)

* * *

Proponents of inter-disciplinary research sometimes seem to believe that the boundaries between the different sciences are no more than unfortunate accidents of history. But the way in which scientific knowledge has been parcelled out over the different disciplines is not so accidental at all: what may constitute the area of a viable scientific discipline has to meet quantitative and qualitative constraints.

Among the quantitative constraints I mention that the area must be small enough so that the major highlights fit in a single person's head; on the other hand it must be large enough to provide intellectual food for at least a lifetime.

Among the qualitative constraints I mention that on the one hand its problems must be sufficiently independent from the rest so that they can be studied in relative isolation from that rest, while on the other hand the area has to have an internal coherence.

It is this last requirement that makes quite clear how the earlier departments of computing science predated computing science itself: to begin with

they were no more than ill-considered cocktails of presumably computer-related topics that happened to be available on campus, eg. some electronics, some numerical analysis, some statistics and economics, some business administration and in the USA some artificial intelligence after a while. The cocktail, composed without coherence, did not taste too good. Forging coherence was one of the first tasks for those that were responsible for carving a niche in which a budding computing science would be viable.

For the sake of the coherence they withdrew from all specific areas of potential computer applications and tried to concentrate on what all those applications ought to have in common; they had done so for the sake of coherence and for the sake of the generality that is required for durability.

Another aspect of that durability was the conscious effort to avoid the training of scientists with a half-life of five years. This implied in particular staying away from everything that could only be given a meaning in close relation to the computing machines currently on the market. For instance, how to live with the idiosyncracies of OS/360 was considered a moving target that was none of computing science's concern.

In summary: the niche was carved, away from specific applications and away from specific machines. And, as time went on, in the same vein away from specific programming languages and operating systems. In the beginning this was done in order to protect the budding science from the volatility of the products from the market place, and when some of these products turned into de facto standards it was done to protect the flourishing science from the stagnation in the market place. Independent of the question whether to regret or to applaud this separation, I want you to understand that for the emergence of computing science as a viable discipline this separation was and still is a conditio sine qua non.

What did computing science do in that splendid isolation? More precisely, what did it do without losing the claim of applicability? Well, it did a lot; much more, in fact, than I can explain in the context of this lecture, but I can give you the flavour.

In the sixties it developed the parsing theory that was needed to raise compiler design above the level of error-prone adhoccery and to turn it into a teachable topic. This was a major achievement: I, for instance, quite distinctly remember how in 1962 those of us who had actually written a compiler were by the others still regarded as a kind of semi-gods. In connection with this achievement I would like to stress that it could never have occurred had we not learned in the mean time how to give a complete formal definition of the syntax of the programming language to be compiled: without that formal definition the compilation problem would have been too ill-defined to exist. Automata theory and complexity theory were developed to give essential and quantitative bounds on what can be computed at all; again, at the heart of these theories lies a very formal postulate as to what computing is, a postulate without which those theories cannot exist. For the sake of operating system design, the problem of process synchronization was posed and solved and the first theorems about the absence of deadlock were proved; again, a formal definition of the phenomenon intuitively known as deadlock was the first prerequisite for that achievement.

In the seventies the attention shifted from syntax to semantics, to begin with for deterministic sequential programs, but shortly thereafter including nondeterminacy and concurrency as well. I shall not try to describe the various

activities in any detail: they ranged from providing a model for the typed lambda-calculus to the development of programs by means of semantics-preserving program-transformations. It was during that decade that programs became mathematical objects in their own right. The most concise way of capturing the change in attitude is probably to remark that, while formerly it had been the task of the programs to instruct our machines, it had now become the task of the machines to execute our programs. Program verification and program design were developed as branches of formal mathematics, with the result that it was no longer an act of irresponsibility to publish a program without having tried it on a computer.

Well, this has been a far from complete overview of how computing emerged as a science, and I apologize to all contributers whose work has remained unmentioned. But I do hope that it has been sufficiently complete and evocative to have given you the flavour of the quintessence of the discipline. It became a fascinating discipline because the separation between "pure" and "applied", which is traditional in so many others, got completely blurred and largely lost its significance. The luxury of working in an environment in which the distinction between pure and applied science is meaningless is probably a fringe benefit of the fact that the general purpose computer really deserves the epithet "general purpose".

It has all the flavours of pure mathematics, being more formal than most other branches of mathematics. It cannot escape being so formal since any programming language, by the sheer virtue of being mechanically interpretable, represents a formal system of some sort. At the same time it has the full flavour of applied mathematics because the sheer capacity of modern computers provides such an opportunity for confusion that its methods are indispensable if we prefer not to get trapped in the complexities of our own making.

Learning how not to get caught in the complexities of our own making, keeping things sufficiently simple and learning how to reason sufficiently effectively about the designs we are considering, all these have become central issues of computing science. This too was recognized more than a decade ago, when "separation of concerns" became a catch phrase of programming methodology.

Please notice that the way in which computing science carved itself its niche was, all by itself, an example of successful "separation of concerns". Whenever we invest a lot of mental energy in the careful design of any discrete system, we do so for more than just the fun of it: we always hope that the product of our efforts will be used for the benefit of others. We hope that it will satisfy a need, will meet the expectations and will please its users. In the prescientific period of system design, the unformalized notion of "user satisfaction" was for a time the only accepted quality criterion for software.

The shortcoming of "user satisfaction" as a quality criterion is that it is not a technical notion: it provides no technical guidance to the designer and, besides that, can be achieved by other than technical means, such as heavy advertising and brainwashing. Science can do nothing with it, at any rate preciously little. The concerns had to be separated, and that is where the functional specification entered the picture.

The role of a formal functional specification is simply to act as a logical firewall between two completely different concerns, known under the names of "the pleasantness problem" and "the correctness problem". The pleasantness problem concerns the question whether a system meeting such-and-such a formal functional specification would satisfy our needs, meet our expectations and fulfil our hopes. The correctness problem concerns the question whether a given design meets such-and-such a formal functional specification.

The logical firewall was necessary so as to create the correctness problem for computing science to tackle: it isolates computing science's well-carved niche from the pleasantness problem to which science has little to contribute. Please note that I did not say that the one problem is more important than the other; after all, no chain is stronger than its weakest link. What I said was that the correctness problem represented that part that we managed to bring inside Bondi's "thin boundary layer" where the conscious application of the techniques of scientific thought can avail, whereas the unformalized pleasantness problem intrinsically lies outside the domain of science.

You may have all sorts of problems, ranging from a dangerous intersection to whole generations of your population feeling threatened in their very existence. But science never solves your problems, it only solves its own ones, and the decision whether you accept the solution to the formal problem science posed to itself as a possible solution to your problem as well is unavoidably yours. In the same vein: science never provides a model for reality, it only constructs its own theory, and the question whether you accept your perception of reality as a sufficiently truthful model for that theory, well, that's your problem.... Truth and reality are no longer scientific notions and the scientist leaves talking about them to the philosophers, the prophets and the poets.

Seen this way, the rôle of science is quite limited, so disappointingly limited in fact, that many prefer to close their eyes to the limitations of science. Stressing these limitations was never very popular in the scientific community, among other things because it raises the question why society should tolerate scientists at all. This is no joke: we all know that if today a society decided to expel its scientists, it would not be the first one to do so...

And now that science has "gone public", so to speak, it is not very popular among the public at large either. People have always had ambivalent feelings towards technology, and the more powerful the technologies people are confronted with, the more dramatic the dilemma of that ambivalence. They feel more threatened by technology than ever before; at the same time their hope of the salvaging power of science and technology is becoming more and more unbridled. In the good old days of the traditional quack, his ointment only needed to cure all ills, his Elixir only needed to give you eternal youth, and from the famous Philosophers' Stone only the modest trick of making gold was required. Those were the good old days, when the quest for the Philosophers' Stone was still in its infancy.

But in these advanced electronic days, the Philosophers' Stone has acquired completely new dimensions. An achievement still in the traditional vein of gold-making is that thanks to robotics eventually all countries of the world will have a positive balance of trade. It will solve the problems of production and unemployment. It will give central government the power to fight crime and corruption, while the ubiquitous micro will be the safeguard of democracy. Teaching machines will rejuvenate the educational process while calculators, automatic spelling correctors and intelligent machines in general will make most education superfluous. All state secrets will be absolutely protected by infallible encryption; powerful decoding schemes will enable us to break all codes. Weapons and defence systems will be equally effective. But, most important of all, if we don't know what to do, we shall have our decisions "supported", our management will be informed and our information will be managed, our design will be as aided as our intelligence will be amplified, and without any special training everybody, really everybody, even managers and generals, will have all the canned expertise they need at their fingertips. The great novelty of today's Philosophers' Stone is that you can delegate your responsibilities to it.

So much for the blatant nonsense. Now it could be argued that in my unwillingness to compromise I have taken an extreme position; I know that quite a number of reasonable and respectable colleagues in the field would judge the limits as I have drawn them for computing science impractically narrow. They would point to all sorts of systems of great potential utility that would fall outside my very strict boundaries. They would argue, firstly, that currently such systems are designed without much of a formal functional specification, and, secondly, that we would not know how to give such a specification, even if we wished to do so. One example is provided by libraries of numerical routines for which neither the limits of their applicability nor the accuracy of the results have been clearly stated. Another example could be a system for optical character recognition as might be used in mail sorting, at least if delivered without a precise statement of which characters will be flawlessly recognized. True enough, but allow me to point out a few things.

Firstly, let me recall that I refused to grade the pleasantness problem and the correctness problem in relative importance, in other words that there is nothing intrinsically wrong with an unscientific project: it is only unscientific. Secondly, that these are projects in which apparently computing science's niche does not fit; well, if that's the case, computing science cannot contribute to them and had better leave them alone. Why should computer applications to which science cannot contribute be ruled out? I don't think we should be bothered about that. It is my firm conviction that we grossly underestimate the cultural significance of computers as long as we judge them primarily as tools, because I expect them to have a much profounder influence in their capacity of intellectual challenge.

My overriding concern, however, is that such projects exploit only part of the computer's characteristics and that they lead to rather unspecified products in which the computer's other characteristics have disappeared. The numerical routines exploit the number crunching abilities, optical character recognition exploits the storage capacity and the flexibility, but in both cases the final product is no longer an automaton with precisely stated properties. Hence the resulting product is no longer a machine in the sense of computing science and using it is like doing mathematics without axioms. Such numerical routines can only be used in a context where the answer does not matter or the numerical analyst using them has other means of verifying the answers; such optical character recognition systems can only be used in circumstances in which it does not matter if, say, a fraction of the letters is initially sent to the wrong destination. We can no longer rely on such systems as if they were automata, and we have to reject them in any closed loop.

For many, such a conclusion is unpalatable, and as a result there is a school of thought --or, if you prefer: a school of non-thought-- that proposes that the situation is not that serious, that we should not be that rigid, that engineers have always allowed their components to fail occasionally, that striving for perfection quickly becomes counter-productive, and that we had better learn how to live in the real world with systems and subsystems that usually do what we expect them to do. It is a seductive proposal. Wouldn't it be nice to achieve excellence without having to strive for perfection? (This the more so, since in a heavy populist society, the latter is socially unacceptable.) It is the stance now taken under the banner of "software engineering".

But what proponents revealingly refer to as "the software engineering movement" is beset by a few unsurmountable internal contradictions.

One of them is the dogma that striving for perfection is counterproductive in the sense that it would make software development much too expensive. But what are the main causes of the soaring costs of software development? A major cost, in terms of both manpower and unforeseen delays, is debugging, and one can save a lot by investing more in preventing the bugs from entering the design in the first place. Since the errors are so expensive, in general the high-quality design is also by far the cheaper. Another major cause is that many systems are built on shifting foundations in the sense that the underlying software of operating systems and compilers is too shaky to be stable, with the result that each new release of that underlying software requires possibly extensive adaptation of what has been built on top of it. Finally, many of the tools the programmer is supposed to work with are so poorly documented that they force him to find out by experiment what they might be able to do for him. Since these experiments can be pretty expensive and time-consuming and --inductive reasoning being what it is-- an educated guess is the best the poor programmer can hope for, the poor programmer is really in a miserable position. So here you see three major sources of cost explosion traced down to someone's assumption that striving for perfection is counterproductive!

Another contradiction of the software engineering movement surfaces as soon as a software engineer asks himself how he could become a better one, how the art, craft or practice of software engineering could be improved. He will immediately discover that he has to resort to the discipline he has rejected. Having accepted as its charter "How to program if you can't.", software engineering has manoeuvred itself into an impossible position, a position computing science can avoid only by refusing to compromise, by sticking to its own formal discipline, and by not pretending to be more.

A moment ago I mentioned poor documentation of a system as an intrinsic limitation on the reliability with which the system can be used mechanically in a larger context. This is the place to point out that hiring a technical writer is rarely a solution; the act is usually not much more than an admission that the system's designers are in some sense functionally illiterate. The common situation is that even an army of technical writers could not do the job because the system has grown so complex that it defies accurate description.

A striking example of this phenomenon has recently been provided by Ada. If Ada is going to provide a standard, that standard had better be unambiguously documented. At least two groups have had a go at it; both efforts resulted in formal texts of about 600 pages, i.e. many times longer than needed to ensure the impossibility of ever establishing firmly that the two documents define the same programming language. The fault of the patent unmanageability of these two documents lies neither with the two groups that composed them, nor with the formalisms they have adopted, but entirely with the language itself: only by not providing a formal definition themselves, could its designers hide that they were proposing an unmanageable monstrum. That Ada will decrease the pain of programming and increase the trustworthiness of our designs to within acceptable limits is one of those fictions for which a military education is needed in order to believe in it. The best thing I can report on this front are persistent rumours that even a military education does not suffice to maintain faith in the effectiveness of this Stone of the Philosophers. I have mentioned Ada explicitly because it provides a perfect illustration of what I alluded to at the beginning of this talk: its adoption was a political process in which computing science, whose warnings were viewed as a nuisance, was not allowed to have any influence, and, consequently, just maintaining one's well-considered doubts becomes, beyond one's control and intentions, a politically loaded act.

I haven't talked about artificial intelligence yet. Well, this topic is fraught with another political complication as it has become part of the transatlantic controversy: the topic never really caught on in Europe. For the first two decades after the war, a simple financial explanation sufficed. Artificial intelligence was expensive and Europe was poor; in addition, artificial intelligence was almost exclusively financed by the DoD, which focussed its funding --I won't say: support-- on American research. But the financial explanation cannot be the whole story, because when Europe was rich enough to fund its own research in artificial intelligence, the topic still did not catch on. Neither did the other soft sciences really.

My conclusion is that it is but one aspect of a broader cultural difference. The European mind maintains a greater distinction between Man and Machine and has lower expectations from both of them. It is less inclined to describe the human psyche in mechanistic terms; it is also less inclined to describe inanimate machinery in anthropomorphic terminology; consequently, it considers the question whether machines can think as relevant as the question whether submarines can swim. Its society is evidently less gadget-ridden, in part because it expects much less from gadgetry, certainly not salvation. Conversely, the goal of mimicking human reasoning is apt to evoke the comment "Couldn't you try to copy something better?"

Usually I don't need to talk about artificial intelligence as it is only a specific area of potential machine application and as such outside the scope of computing science proper. I have to do it, however, as soon as it is proposed that, by applying AI techniques, machines will solve the software problems we don't know how to solve ourselves. Our first reaction to the Fifth Generation Project was a sigh of relief, in the style of "Well, if the Japanese industry tries to cash in on AI, that will take care of the Japanese competition.". Within a week or so came the sad realization that the Western world would probably lack the backbone needed for not joining that bandwagon.

Indeed it joined the bandwagon and, consequently, we have again the popularity of "Wouldn't it be nice if our machines were smart enough to allow programming in natural language?". Well, natural languages are most suitable for their original purposes, viz. to be ambiguous in, to tell jokes in and to make love in, but most unsuitable for any form of even mildly sophisticated precision. And if you don't believe that, either try to read a modern legal document and you will immediately see how the need for precision has created a most unnatural language, called "legalese", or try to read one of Euclid's original verbal proofs (preferably in Greek). That should cure you, and should make you realize that formalisms have not been introduced to make things difficult, but to make things possible. And if, after that, you still believe that we express ourselves most easily in our native tongues, you will be sentenced to the reading of five student essays. The problem with the "smart" machines is the same as we had with all the programming language "features": each next layer of "user-friendliness" blurs the specification and thereby makes the system more risky to use.

One final issue. It is not the topic of my talk but, in these political days, if I don't raise it myself, it will be raised in the discussion: what about the software required for SDI, better known as Star Wars? Well, I am sure I could not design it to my satisfaction.

I thank you for your attention.