Copyright Notice

On the role of scientific thought.

Essentially, this essay contains nothing new, on the contrary: its subject matter is so old, that sometimes it seems forgotten. It is written in an effort to undo some of the more common misunderstandings that I encounter (nearly daily) in my professional world of computing scientists, programmers, computer users and computer designers, and even colleagues engaged in educational politics. The decision to write this essay now was taken because I suddenly realized that my confrontation with this same pattern of misunderstanding was becoming a regularly itself repeating occurrence.

Whether the misappreciation of the proper role of scientific thought that I observe within the "computing community" is a phenomenon that is specific for the computing community, or whether it also a current phenomenon in other disciplines, is not for me to judge. One thing seems certain: in the computing community itself we can find enough historical explanation, and we don't need to look for outside influences when we try to understand how the phenomenon came about. (This is not meant to say, that those outside influences have been absent!)

As we shall see in a moment, the adjective "scientific" when used in the expression "scientific thought" more refers to a way of thinking than to what are the thoughts about: to use the Latin expressions: it refers to the "quo modo" rather than to the "quod". This partly explains why the tradition of scientific thought has only been imported into the computing world to such a limited extent by the many pioneers who immigrated in the early days from other scientific disciplines. The early academics who became involved with computers all had had their training in other scientific disciplines and many of them were quite able to practice "scientific thought" in their original field of intellectual activity. But for a great number of them, that had been the only confrontation with scientific thought. As a result, it is understandable that they associated their notion of scientific thought as much with the specific field in which they had practiced it as with a general way of thinking that could (and should!) be transferred to their new field of activity. In addition, many of them must have felt that scientific thought was a luxury that one could afford in the more established disciplines, but not in the intellectual wilderness they now found themselves in. But, as we shall also see in a short while, scientific thought is not a luxury made possible in established scientific disciplines, on the contrary: it has been the tool that has made the establishment of those disciplines possible!

Besides emigrants from other academic fields, the computing world has attracted people from all over the world: businessmen, administrators, operators musicians, painters, unshaped youngsters, you name it, a vast majority of people with no scientific background at all. By their sheer number they form all by themselves already an explanation for the phenomenon.

To introduce the subject, I would like to quote two paragraphs from a letter that I recently wrote to one of my professional friends.

"Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. We know that a program must be correct and we can study

it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained --on the contrary!-- by tackling these various aspects simultaneously. It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focussing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

"I remember walking with Ria when we were engaged --it was near Amsterdam's Central Station-- when I explained to her that I wanted to be glad and happy with my eyes fully open, without fooling myself in the belief that we lived in a pink world: to be happy to be alive in the full knowledge of all misery, our own included...." (End of quotation.)

Scientific thought comprises "intelligent thinking" as described above. A scientific discipline emerges with the --usually rather slow!-- discovery of which aspects can be meaningfully "studied in isolation for the sake of their own consistency", in other words: with the discovery of useful and helpful concepts. Scientific thought comprises in addition the conscious search for the useful and helpful concepts.

The above should make it clear that I want to discuss the role of scientific thought for the sake of its practical value, that I want to explain my pragmatic appreciation of a tool. It is no slip of the pen that the above quotation refers to the "effective ordering of one's thoughts": the efficiency of our thinking processes is what I am talking about. I stress this pragmatic appreciation because I live in a culture in which much confusion has been created by talking about the so-called "academic virtues" (sic!) with moral, ethical, religious and sometimes even political overtones. Such overtones, however, only confuse the issue. (If you so desire, you may observe here scientific thought in action: I do, for instance, not deny political aspects, I would be a fool if I did so! The anti-intellectualistic backlash against "the technocrats" that is so en vogue today, is inspired by a --largely unjustified-- fear for the power of him who really knows how to think and by a --more justified-- fear for the actions of him who erroneously believes to know how to think. These political considerations, however, have nothing to contribute to the technical problem of ordering one's thoughts effectively, and that is the problem that I want to discuss "in isolation, for the sake of its own consistency".)

I intend to describe for your illumination the most common cases in which the "average" computing scientist fails to separate the various concerns; in doing so I hope and trust that my colleagues in the profession do interpret this as an effort to help them, rather than to insult them. For the sake of the non-professional, I shall present the least technical cases first.

One of the concerns, the isolation of which seems most often neglected, is the concern for "general acceptance". (In the world of pure mathematics --with which I have some contacts-- this problem seems to be fairly absent.) The concern itself is quite legitimate. If nobody reads the poems of a poet that wanted to communicate, this poet has failed, at least as a communicating poet. Similarly, many computing scientists don't just solve problems, but

develop tools: theories, techniques, algorithms, software systems and programming languages. And if those, that --they feel-- could profit from their designs, prefer to ignore these inventions and to stick to their own, old, rotten routines, the authors get the miserable feeling of failure. Have they? Yes and no. They can adopt the Galileian attitude: "Nothing becomes true because tenthousand people believe it, nor false because tenthousand people refuse to do so.", and can decide to feel themselves, in splendid isolation, superior to their fellow computer scientists for the rest of their lives. I can deny no inventor who feels underappreciated, such a course of action. I don't recommend it either: the sterile pleasure of being right tends to get stale in the course of a lifetime. If one's aim is to design something useful, one should avoid designing something useless, because unused: in other words, I fully accept "general acceptance" as a legitimate concern. We must, however, be willing to ignore this concern temporarily --for a few days or a few years, depending on what we are undertaking-- for unwillingness to do so will paralyze us.

Some time ago I visited the computing center of a large research laboratory where they were expecting new computing equipment of such a radically different architecture, that my colleagues had concluded that a new programming language was needed for it if the potential concurrency were to be exploited to any appreciable degree. But they got their language design never started because they felt that their product should be so much like FORTRAN that the casual user would hardly notice the difference "for otherwise our users won't accept it". They circumvented the problem of explaining to their user community how the new equipment could be used at best advantage by failing to discover what they should explain. It was a rather depressing visit....

The proper technique is clearly to postpone the concerns for general acceptance until you have reached a result of such a quality that it deserves acceptance. It is the significance of your message that should justify the care that you give to its presentation, it may be its "unusualness" that makes extra care necessary. And, secondly, what is "general"? Has Albert Einstein failed because the Theory of Relativity is too difficult for the average highschool student?

Another separation of concerns that is very commonly neglected is the one between correctness and desirability of a software system. Over the last years I have lectured for all sorts of audiences about the techniques that may assist us in designing programs such that one can prove a priori that they meet their specifications. One of the standard objections raised from the floor is along the following lines: "What you have shown is very nice for the little mathematical examples with which you illustrated the techniques, but we are afraid that they are not applicable in the world of business data processing, where the problems are much harder, because there one always has to work with imperfect and ambiguous specifications." From a logical point of view, this objection is nonsense: if your specifications are contradictory, life is very easy, for then you know that no program will satisfy them, so, make "no program"; if your specifications are ambiguous, the greater the ambiguity, the easier the specifications are to satisfy (if the specifications are absolutely ambiguous, every program will satisfy them!).

Pointing that out, however, seldom satisfies the man who raised the objection. What he meant, of course, was something different. He meant something along the following lines: "We make something with the best of intentions

in the hope of satisfying a need as we understand it, but when our product has been put into action, it does not perform satisfactorily and how are we to discover whether we have correctly made the wrong thing or whether their is just a silly bug somewhere?". The point is that this question is empty as long as the specifications do not define --are not accepted to define by definition-- what the system is supposed to do. It is like asking the judge to settle a business dispute caused by the absence of a contract stating the mutual rights and obligations. It is the sole purpose of the specifications to act as the interface between the system's users and the system's builders. The task of "making a thing satisfying our needs" as a single responsibility is split into two parts "stating the properties of a thing, by virtue of which it would satisfy our needs" and "making a thing guaranteed to have the stated properties". Business data processing systems are sufficiently complicated to require such a separation of concerns and the suggestion that in that part of the computing world "scientific thought is a non-applicable luxury" puts the cart before the horse: the mess they are in has been caused by too much unscientific thought.

But from the above, please don't conclude that unscientific thought is restricted to the business world! In Departments of Computing Science, one of the most common confusions is the one between a program and its execution, between a programming language and its implementation. I always find this very amazing: the whole vocabulary to make the distinction is generally available, and also, the very similar confusion between a computer and its order code, remarkably enough, is quite rare. But it is a deep confusion of long standing. One of the oldest examples is presented by the LISP 1.5 Manual: halfway their description of the programming language LISP, its authors give up and from then onwards try to complement their incomplete language definition by an equally incomplete sketch of a specific implementation. Needless to say, I have not been able to learn LISP from that booklet! I would not worry, if the confusion were restricted to old documents, but, regretfully enough, the confusion is still very popular. At an international summer school in 1973, a very well-known professor of Computing Science made the statement that "ALGOL 60 was a very inefficient language", while what he really meant was that with the equipment available to him, he and his people had not been able to implement ALGOL 60 efficiently. (That is what he meant, he did not mean to say it!) Another fairly well-known professor of computing science has repeatedly argued in public that there is no point in proving the correctness of one's programs written in a higher-level language "because, how do you know that its compiler is correct?". In the motivation of a recent research proposal doubt is cast upon the adequacy of "the axiomatic semantics approach" as it may lead to deductive systems that are "undesirable in that they may not accurately reflect the actual executions of programs". It is like casting doubt on Peano's Axiomatization of the Natural Numbers on the ground that some people make mistakes when they try to do an addition!

On the one hand we have the physical equipment (the imolementation), on the other hand we have the formal system (programming language). It is perhaps a question of taste --I don't believe so-- to whom of the two we give the primacy, that is whether it is the task of the formal system to give an accurate description of (certain aspects of) the physical equipment, or whether it is the task of the physical equipment to provide an accurate model for the formal system --and I prefer the latter--. But under no circumstance we should confuse the two!

I have --I think-- very good reasons for my preference, because if I

cannot appreciate a formal system for the sake of its own consistency, but must view it as description of physical equipment, I could not deal with a programming language that has not been implemented! (And that is, for instance, exactly what a language designer has to do.)

The confusion is perhaps most clearly demonstrated by the often expressed opinion that "one cannot use a programming language that has not been implemented". But this is nonsense, of course one can! One can use any well-defined programming language, whether implemented or not, for writing programs in; it is only when you want to use those programs to evoke computations, that you need an implementation as well. Being well-defined, rather than being implemented, is a programming language's vital characteristic.

The above remarks are no jokes, nor puns, on the contrary: they are pertinent to multi-million-dollar mistakes. They imply for instance that the development projects --erroneously called "research projects"-- aiming at the production of "natural language programming systems" --currently en vogue again-- are chasing their own tails.

Note (which I hate to add, because it is nearly an insult to my readers, whom its inclusion accuses of possible superficiality). I have not said that when considering a programming language, one should not care about its implementability: one had better! But also this concern, no matter how serious, is one we should try to isolate.(End of note.)

In my opening paragraph I also mentioned colleagues engaged in educational politics. The writing of this essay was, as a matter of fact, also promted by a recent study of two Computing Science Curricula at university level. They were from different sides of the Atlantic Ocean, but shockingly similar in two respects: unbelievably elaborate budgets and a total lack of understanding of what constitues a scientific discipline.

A scientific discipline separates a fraction of human knowledge from the rest: we have to do so, because, compared with what could be known, we have very, very small heads. It also separates a fraction of the human abilities from the rest; again, we have to do so, because the maintenance of our non-trivial abilities requires that they are exercised daily and a day --regretfully enough-- has only 24 hours. (This explains, why the capable are always busy.)

But of course, any odd collection of scraps of knowledge and an arbitrary bunch of abilities, both of the proper amount, do not constitute a scientific discipline: for the separation to be meaningful, we have also an internal and an external requirement. The internal requirement is one of coherence: the knowledge must support the abilities and the abilities must enable us to improve the knowledge. The external requirement is one of what I usually call "a thin interface"; the more self-suuporting such an intellectual subuniverse, the less detailed the knowledge that its practitioners need about other areas of human endeavour, the greater its viability. In the terminology of the computing scientist I should perhaps call our scientific disciplines "the natural intellectual modules of our culture". (When the layman asks the computing scientist, what is meant by "Modularization", a reference to the way in which the knowledge in the world has been arranged, is probably the best concise answer.)

In view of the preceding it becomes quite obvious why many earlier efforts to concoct Computing Science Curricula at our universities have been such dismal failures. They were just cocktails! For lack of other ingredients, they tried to combine scraps of knowledge from the most diverse fields that seemed to have some relation to the phenomenon Computer. That the ingredients of the cocktail did not mix into a coherent whole, is not surprising; that the cocktail did not taste too well, is not surprising either.

In those early days, the only alternative was waiting, as for instance still in 1969 urged by Strachey: "I am quite convinced that in fact computing will become a very important science. But at the moment we are in a very primitive state of development; we don't know the basic principles yet and we must learn them first. If universities spend their time teaching the state of the art, they will not discover these principles and that, surely, is what academics should be doing." I could not agree more.

Now, of course, one can argue whether five years later we computing scientists have enough of sufficiently lasting value that can be "studied in isolation, for the sake of its consistency". I think that now we have enough to start, but if you think Strachey's advice still appropriate now, you have my full sympathy.

The two recent(!) curriculum proposals I just referred to, however, presented the old cocktail as if absolutely nothing had happened, and, not as a timid first step, but as the final goal..... And when scientists no longer know, what science is supposed to be about, we are in bad shape. Hence this essay.

30th August 1974
Burroughs
Plataanstraat 5
NUENEN - 4565
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow