On a cultural gap (Draft.)

On the typical university campus, the typical mathematician and the typical computing scientist live in different worlds: they don't know each other or, if they do, they are not on speaking terms. The purpose of this essay is two-fold, viz. to give a historical explanation of this phenomenon and to argue that we should do something about it.

Now, exhorting the world to mend its ways is always a tricky business, for implicit in the exhortation is always the verdict that the world's ways leave plenty room of improvement, a suggestion that is always offensive to the touchy. One way of gilding the pill is to qualify one's sentences by all sorts of softeners, such as "typical", "in general", "on the average", "usually", "not uncommonly". For brevity's sake I shall not do so. (A second reason for not doing so is that I do not really believe in the effectiveness of the softeners with which one could lard one's text: those who wish to take offence will anyhow succeed in doing so.)

At the heart of my historical explanation lies the thesis that when, now four decades ago, the automatic electronic computer was sprung on us, we were not ready for it and that thereafter - people and computers being what they are - wide-spread

confusion about what had happened was unavoidable.

By far the most common way in which we deal with something new is by trying to relate the novelty to what is familiar from past experience: we think in terms of analogies and metaphors. As long as history evolves along smooth lines, we get away with that technique. It is obvious, however, that that technique breaks down whenever we are suddenly faced with something so radically different from what we have experienced before that all analogies, being intrinsically too shallow, are more confusing than helpful. But such radical novelties are precisely the things technology can confront us with; the automatic computer was one of them. (From the same era, the pill and the atom bomb were two others.) For coming to grips with radical novelty there is only one way, i.e. consciously trying not to relate it to one's accidental past but to appreciate it for its internal structure. The latter way is far less popular than the former one as it requires hard thinking.

In short, being the radical novelty they were, computers were bound to cause confusion. In the world of computing – we did not talk about computing science yet – an accident of history caused further obfuscation. As I said earlier, we were not ready for computers, and now I may add that

the unprepared included the electronic engineers that were supposed to design, build, and maintain them. The job was actually beyond the electronic technology of the day, and as a result the question of how to get and keep the physical equipment more or less in working condition became in the early years the all-overriding concern.

We <u>now</u> know that electronic technology has no more to contribute to computing than the physical equipment. We <u>now</u> know that a programmable computer is no more and no less than an extremely handy device for realizing any conceivable mechanism without changing a single wire, and that the core challenge for computing science is a conceptual one, viz. what (abstract) mechanisms we can conceive without getting lost in the complexities of our own making.

But in the mean time, the harm was done: the topic became known as "computer science" — which, actually, is like referring to surgery as "knife science" — and it was firmly implanted in people's minds that computing science is about machines and their periferal equipment. Quod non. (These days I cannot enter a doctor's, dentist's, or lawyer's office without being asked my advice about their office computer. When I then tell them that I am totally uninformed as to

what hard- and software products the market currently offers, their faces invariably get very puzzled.) The machine-oriented image of computing science, no matter how obsolete and distorted in the mean time, lingers on. Needless to say, it can only contribute to hiding for the mathematical community that a major part of computing science is by now a -be it perhaps somewhat unusual- mathematical discipline.

When, after the first ten years, computers became available as industrial products, it was a commercial imperative for the computer industry to dissociate their products as far as possible from any form of mathematics, the latter being viewed as the pinnacle of "user-unfriendliness". Its sales force accordingly brainwashed the public, including the computing scientists and mathematicians alike. So much for the external influences that helped form computing science's public image.

As time went by, we got people using machines and on campus we got people involved in what was -timidly at first- called "computing science". Did they promote of their activity an image that would appeal to the orderly and scientific mind that the mathematician regards as his specialty? Certainly in the beginning definitely not.

To begin with, they institutionalized the habit of referring to machines and their components and later to programs in a strongly anthropomorphic terminology. This must have put off many of their mathematical colleagues (certainly those who had learned to interpret the heavy use of anthropomorphisms as a sure symptom of professional immaturity). Though the underlying metaphors have long out-lived their potential usefulness and are now more a handicap than a help, the habit of anthropomorphizing equipment regrettably persists in the computing community until this very day.

Furthermore, the early scientists involved in computing came - of necessity - from other disciplines: they had mostly been trained as physicist, chemist or crystallographer (with an occasional astronomer and meteorologist). They were the machine users of the first hour. But for reasons I have failed to fathom, the vast majority of them have failed to transfer their scientific quality standards to the programming that became their major occupation: as soon as programming is concerned, otherwise respectable scientists suddenly accept to live by the laws of the logical jungle. (To this very day we find physicists willing to parade as computing experts, and that merely on the strength of having burnt up so much processing power.) A generation of "scientific" machine users has approached the programming task more as solving

a puzzle posed by the machine's manufacturer than as an activity worthy of the techniques of scientific thought. Their logical catch-as-catch can must have been repulsive to the orderly mind that -as said- the mathematician regards as his specialty. So much for their contribution to computing's image.

How did the mathematician react to the advent of computers? The simplest answer is: "Not." . As it takes for new scientific developments about half a century to become widely accepted common property, our mathematician's values and prejudices had their roots in the late 19th, early 20th century. He was full of analysis, loved the continuum and the complex plane and considered infinity as a prerequisite for mathematical depth. How in the world could computing have anything to do with him? (In the sixties, a justly famous mathematician, then Chairman of the Department at Leyden University, still thought he could declare that "being a mathematician, he knew of course nothing about computers" without making a fool of himself.) If he acknowledged the existence of computers at all, he viewed them as number crunchers, of possible use as a tool for his colleague in numerical analysis -if he had one-. For numerical analysis, of which he knew very little, he had at best a mild contempt. So he disregarded computing completely, hardened in that attitude when he noticed that machines were primarily used for

business administration, which was a trivial pursuit anyhow.

Occasionally his peace of mind was slightly disturbed when a colleague in number theory or combinatorics proved a theorem for all n by proving it himself for all n exceeding some large N and having one of those machines check it for all smaller values. He quickly restored his peace of mind by frowning upon the practice. Finally he settled on the compromise that a computer might occasionally be a useful tool, but then only for the more repetitious and boring part of the work, but since all computations were essentially finite and all finite problems were in principle trivial, why bother?

That way in which the mathematician soothed himself was, of course, flawed and as time went by he grudgingly admitted that there might be circumstances under which it might be untenable to lump all finite problems together under the category "trivial" and we should admit the subcategory "finite, yes, but much too big if you really want to do it".

There is, however, a much deeper flaw. One day our mathematician concocted an argument that made him feel utterly respectable for it was full of transfinite ordinals and more of such good things.

But when he proudly showed it to his colleague, what did he show: infinite sets or ..... just a finite argument? And this, of course, raises the question what mathematics is about: is it still -The Concise Oxford Dictionary, 6th Edition, 1976- the "abstract science of space, number, and quantity" or is it more "the art and science of effective reasoning"? I think we have here in a nutshell the full profundity of the cultural gap: the mathematician opts for the Oxford definition, the computing scientist for the latter alternative.

Let me try to sketch, by way of explanation of that preference of the computing scientist, what computing science came to be about. Computing science is not about computers, computing science is not about how to use computers in specific areas of potential application, computing science is about how to solve, with or without machines, the (scientific) problems posed by the existence of computers.

As said before, we were unprepared for the advent of the automatic computer, and one of the things we were unprepared for was the shocking discovery -made as soon as the machines became available- that we could not program! All but the most simple programs turned out to be -and remain!- full of bugs.

Understandibly, the blame was put on the way the programmer writing in machine code had to express himself: an unfamiliar notation, the understanding of which required a complete awareness of all the idiosyncrasies of the machine in question. In response to our difficulties, what is now known as "higher-level programming languages" came into being. They offered the programmer a more familiar and more suitable notation that shielded him from specific machine characteristics and the implementation of which relieved him from a number of clerical burdens. They embodied a significant improvement that allowed a change of attitude: had it formerly been the task of the programs to instruct our machines, now it became the task of the machines to execute our programs.

With that change of emphasis, we got our second shock: shielded from machine characteristics and relieved from all sorts of clerical burdens, we still could not program! Slowly it dawned upon us that we could not continue to put the blame on the mechanisms available to us, because the major bottle-neck had become our inability to reason sufficiently effectively about what we were trying to conceive. With that recognition "programming methodology" became an area of explicit scientific concern.

From concerns such as "How do we best structure

a program so as to facilitate our reasoning about it as much as possible?" it was only a short way to "How do we most effectively disentangle complicated arguments?" and, for instance, "modularity" became one of the catchwords. By the end of the sixties it was generally felt by the computing scientists engaged in this kind of methodological investigations that the closest familiar analogue to a well-engineered, sophisticated program was probably an equally well-engineered, sophisticated mathematical theory. Despite that feeling it took quite a few more years before they realized that their efforts had probably be best understood as part of mathematical methodology in general.

In retrospect, that slowness of the computing scientist in recognizing the intrinsically mathematical nature of his work is understandable: he was heading for a kind of mathematics of a flavour different from what the mathematician was wont to do: for the computing scientist, formal techniques would play a much more important rôle than they had done so far for the mathematician.

For the predominance of formal techniques in the computing scientist's work, we can discern -again in retrospect!- three reasons, viz. a practical one, a circumstancial one, and a cultural one.

The practical reason was that formal techniques are indispensable for dealing in a sufficiently trustworthy manner with the type of complexities a program designer has to control.

The circumstancial reason was that any programming language represents by the very fact of its mechanical interpretability a formal system of some sort.

The cultural reason was that the computer scientist regards – "by nature", so to speak – symbol manipulation as mechanical processing of uninterpreted formulae (regardless of whether he actually intends to mechanize parts of the process).

Part of the significance of what has happened has been seen by some of the logicians. P. Martin-Löf – see [0], to which I refer for further references – wrote for instance:

> "It [=the creation of high level languages of a sufficiently clean logical structure] has made programming an activity akin in rigour and beauty to that of proving mathematical theorems. (This analogy is actually exact in a sense that will become clear later.)"

and

> "In fact, I do not think that the search for high level programming languages that are more

and more satisfactory from a logical point of view can stop short of anything but a language in which (constructive) mathematics can be adequately expressed." .

The book [0], from which the above quotations are taken, has been given as motto

"'It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and for mathematical elegance.'
(J. McCarthy, 1967)"

The earlier quotations illustrate that McCarthy's hope is in the process of being fulfilled; his second sentence, however, captures the predicament of the computing scientist. The logician primarily studies formal methods in an effort of capturing how the mathemation does or should behave himself: he more ponders about those methods than that he uses them himself. The mathematician knows of the existence of mathematical logic but never uses it. (With a simple test I have actually established that, up till now, for the mathematical community George Boole has lived in vain: the fraction of mathematicians well-versed in the

propositional calculus is absolutely negligible.) He definitely associates logic with the purer side of mathematics. In our history, the computing scientist seems to be the first to be asked to apply formal logic on a grandiose scale and to make formal techniques an integral part of his daily reasoning; in a way, circumstances have forced him to become more mathematical than the Pope is Roman Catholic.

My expectation – but I have often been accused of naive optimism! – goes further than McCarthy's hope. I expect formal techniques to have in the next century a most profound influence on mathematics in general. Currently, the mathematicians are an informal lot, a state of affairs defended by the remark that a more formal treatment would make the topic boring, tedious, laborious, repetitious and what have you. The defence may seem valid, but what else can you expect in a world that has never seriously tried to apply formal techniques on a sizeable scale with elegant effectiveness? A convincing defence it is not.

From both sides, closing the gap will be most likely be experienced as a painful process. For computing scientists with an experimental bent it will be unsettling to enter a culture that would prefer their experiments to be superfluous; for the traditional mathematician it will be unsettling to learn that his time-honoured way of working is just no

longer adequate. But that is precisely why I want the gap between the computing scientist and the mathematician to be closed: neither of them should deny themselves the challenge that is implied.

[0] Hoare, C.A.R. and Shepherdson, J.C. (Eds.), Mathematical Logic and Programming Languages, Prentice-Hall International London, 1985
ISBN 0-13-561465-1

Austin, 24 March 1985

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