# Why American Computing Science seems incurable

Discussing how well academic Computing Science fares, we have, in the USA, to pay more attention to the Computing Industry than in other countries, and we have to do so for two reasons: firstly, compared to other countries, there is hardly a fence that separates the American university campus from the society that surrounds it, secondly, most, if not almost all, of the world's Computing Industry is concentrated in the USA. Consequently, opinions and prejudices prevailing in the American Computing Industry are here for academic Computing Science an undeniable force, be it driving or paralyzing.

For the sake of the stability of the enterprise, the ideal of its manager is an organization that is as independent as possible of specific abilities of individual employees. The predominance of this ideal is a well-documented, international phenomenon; the ideal itself predates the high-technology industry, in which it could very well be inappropriate, and has discouraged the industrial employment of scientists, in particular of the brilliant and original ones.

The American situation is aggravated by a total lack of faith in its educational system and a deep-rooted mistrust of intellectuals.

Aside How typically American this mistrust is, is illustrated by the fact that the American word "egghead" has no Dutch translation. I looked it up in my Webster's New Collegiate Dictionary (1973), which in its quotation fully captures the untranslatable connotation:

> "egghead n: INTELLECTUAL, HIGHBROW
> < practical men who disdain the schemes and dreams of ~s — W.L. Miller > ".

By Dutch standards, Miller's quotation is an unambiguous disqualification of "practical men". (End of Aside.)

Accordingly, prevailing industrial attitude exerts a strong pressure on the University not to indulge in such hobbies as scientific education, but to confine itself to vocational training of some sort or another.

But in the case of Computing Science, it is not only the teaching that is influenced by the industrial prejudices, research suffers as well. I remember, for instance, a so-called "faculty candidate" who talked about his Ph.D. work — they usually do —, which was

a system for the "parallelization" of a modest class of Fortran programs. The speaker did not claim that he had gained any new insights or that you could learn anything interesting from studying his thesis, and the sole justification of his work was to be found in his final product, viz. his Fortran program parallelizing software. And this software was very important because (i) there were thousands of Fortran programs "out there", and (ii) the system was fully automatic, an absolute requirement for industrial acceptance. (I am not making this up!)

I thought that the main criterion by which to judge our academic research is how it improves our teachable material, but this poor bloke had adopted "industrial acceptance" as quality criterion, and as a result his gadget worked in such a small set of circumstances that its main feature became that one could apply it unthinkingly.

There are other instances of how research suffers from industrial pressure.

Being a better programmer means being able to design more effective and trustworthy programs and knowing how to do that efficiently. It is about not wasting storage cells or machine cycles and about avoiding those complexities that increase the number of reasoning steps needed to keep the design under strict intellectual control. What is needed to achieve this goal, I can only describe as improving one's mathematical skills, where I use mathematics in the sense of "the art and science of effective reasoning". As a matter of fact, the challenges of designing high-quality programs and of designing high-quality proofs are very similar, so similar that I am no longer able to distinguish between the two: I see no meaningful difference between programming methodology and mathematical methodology in general. The long and the short of it is that the computer's ubiquity has made the ability to apply mathematical method more important than ever.

In a cruel twist of history, however, American society has chosen precisely the 20th Century to become more and more a-mathematical (a phenomenon, for instance observed —and bemoaned— by Morris Kline).

and we have reached the paradoxical state that, of all so-called developed nations, the USA is the most dependent on programmed computers and intellectually the worst equipped to be so. The suggestion that the programming problem could be amenable to mathematical treatment is, if heard at all, instantaneously rejected as being totally unrealistic.

As a result, Program Design is prevented from becoming a subdiscipline of Computing Science. There is considerable concern for correctness, but almost all of it has been directed towards a posteriori program verification because, again, that more readily appeals to the dream of complete automation. But, of course, many - I included - regard a posteriori verification as putting the cart before the horse because the whole procedure of programming first and verifying later raises the burning question where the verifiable program comes from. If the latter has been derived, verification is no more than checking the derivation. And in the mean time, programming methodology - renamed "software engineering" - has become the happy hunting-ground for

the gurus and the quacks.

Deprived of what is generally considered computing's core challenge, American Computing Science is the big loser, and we cannot blame the universities, for when the industry most in need of their scientific assistance is unable to face that they are in a high-technology business, even the best university is powerless. Universities should be more enlightened than their environments, and they can, but not much (that is, not openly). In the current political climate, it is unlikely that things will improve soon; in the near future we shall have to live with the superstition that also programming is "so easy that even a Republican can do it".

Austin, 26 August 1995

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