Can computing science save the computer industry?

Yeah, can computing science save the computing industry? Very good question. It has at least the virtue of being controversial. Departments of computing science all over the world work (and are funded) under the tacit assumption that the answer is of course affirmative. (In fact, the assumption is so tacit in those circles that the question is never raised there.) The computer industry, however, operates under the equally tacit assumption that — thank the Lord! — computing science has nothing to contribute to its business, which — as everyone knows — is primarily a matter of marketing, of timely inventing the proper slogan in name of which the next castle-in-the-air can be presented as solid real-estate.

As long as things seemed to work out quite well, no one was eager to raise the question. But things have changed. The industry is in grave trouble, because development and maintenance costs are soaring and the public at large is getting disillusioned with its products. The university departments are in trouble too: the computing boom has decreased their staff and increased their teaching load, and as far as computing science is concerned, most faculty members have no time for it anymore and most students are no longer interested in it. So let us pose the question.

* * *

To begin with a number of observations that should be taken into consideration.

(A) The core of the challenge that computing science has accepted is how not to get confused by the complexities of our own making. In passing we note that in those areas in which computing science has attacked this problem explicitly (e.g. the development of programs) it has been noticeably successful.

(B) I am waiting for the stage -if it has not been reached yet- that all sorts of products of the computer industry -e.g. microprocessors- are (in some sense) cheaper than their documentation; giving a concise and complete functional specification -traditionally produced months, if not years, after the fact, if at all- is regarded as a major (if not impossible) task. In short, even in stand-alone products the computing industry has not been able to keep its products sufficiently clean and simple.

(C) A major source of complexity is provided by the so-called "de facto standards", be it programming languages (Fortran, PL/I, Cobol, Ada), data base systems, or communication protocols. Wherever computing science has made significant progress -e.g. programming- ignoring those de facto standards has each time been a necessary condition. The de facto standards are, indeed, so terrible -all of them!- that if the computer industry persists in adhering to them -for the so-called

"sound commercial reasons"— computing science will be unable to save it.

(D) The computer industry's way of speaking about its customers is very condescending: the user - sometimes strengthened as "the end user" - is treated as a moron - e.g. "making our systems so user-friendly that even housewives can use them"—. It even talks in derogatory terms about its own employees. In fact, their postulated lack of education or their education resistance, their postulated inability to write clearly and concisely, etc. are often used by management as a pretext for not trying to do a better job. As long as that pretext is used, computing science will be unable to save the computer industry.

Aside Having your designs made by illiterate engineers and then described by technical writers, is an unworkable division of labour: the designer thus misses the feedback he needs. The lesson learned from programming language design - a lesson grossly ignored in the Ada development process! - is that precise definition as a design principle acts as an indispensable early-warning system for complexity inadvertently creeping in. (End of Aside.)

(E) The industry's propensity to look for better "tools" is a reflection of the condenscending style mentioned under (D). It is a reflection of the

management philosophy that, for the sake of continuity and invulnerability, organizations should rely as little as possible on the specific abilities of individual employees. We should not forget that this management tradition is significantly older than the high-technology computer industry and, hence, could now very well be profoundly inadequate. As long as managers regard program development as a production process - and they do! - and express "programmer productivity" in terms of "number of lines of code produced per day" - and they do! - their mental categories are indeed inadequate. At my lectures for programmers on programming, all over the world the overwhelming reaction has been "What a pity our manager is not here!". My conclusion is that if the computer industry is unable to refresh - by education or by replacement - its by-and-large technically incompetent management, computing science will not be able to save the computing industry.

Some of my readers may think the conditions mentioned under (C), (D), and (E) so "unrealistic" that they have already concluded that computing science cannot save the computing industry. Those readers don't need to read any further. The following is written under the assumption that one day the computer industry will be wise or desperate enough to see that drastic improvement without drastic change is unrealistic.

(F) We continue our explorations. The prime paradigma of the pragmatic designer is known as "poor man's induction", i.e. he believes in his design as long as "it works", i.e. until faced with evidence to the contrary. (He will then "fix the design".) The scientific designer, however, believes in his design because he understands why it will work under all circumstances. The transition from pragmatic to scientific design would indeed be a drastic change within the computer industry.

(G) Conversely, without the transition from pragmatic to scientific design, the industry's intellectual resources will continue to be usurped by "maintenance" of systems that should never have been designed the way they are in the first place.

(H) Combining (F) and (G), we see a prime responsibility for (industrial) computing science research: making the transition from pragmatic to scientific design technically possible. I call this "a prime responsibility" in the sense -see (G)- that this goal is so crucial that, if not attainable, we must conclude as yet that, regrettably, computing science is unable to save the computing industry.

(I) The prime responsibility revealed under (H) is a mathematical challenge, of a scope radically different from the mathematical challenges accepted

so far. The currently prevailing mathematical traditions, rooted in the last centuries, are, though time-honoured, no longer adequate and we now have to learn how to reason an order of magnitude more effectively than we are used to do. This conclusion should turn "mathematical methodology" into a prime issue for computing science research (and should convince the industry that mathematical elegance is not a dispensable luxury but the decisive factor between technical success and technical failure of its products).

(J) There are several reasons to believe that research in mathematical methodology is a highly rewarding exercise. Firstly, it has been a neglected topic: it is no accident that the COD defines mathematics in 1976 as "abstract science of space, number, and quantity" rather than as "the art and science of effective reasoning". Secondly, the calling of logic – "Philosophy's Handmaiden" – has traditionaly been the codification of how people think – more accurately: how people should think if they don't make mistakes –; despite the emergence of mathematical logic, the use of logical formalisms with the intent of freeing us from the shackles of our natural languages is still in its infancy. Thirdly, if there is an analogy between programming and doing mathematics – and I think that analogy is profound –, it is reasonable to hope that research on mathematical methodology will be as rewarding as research on programming methodology has been. (It could be argued that

mathematics today is as much paralyzed by the concept of "the average mathematician" as programming has been by that of "the average programmer".)

(K) For purely technical reasons it is to be expected that more and more mathematical reasoning will will be in terms of formal calculations, i.e. in the form of symbol manipulations applied to uninterpreted formulae. This will take place in the name of effectiveness, independent of the circumstance that the formal calculations should become amenable to mechanical verification. Such tools should only be defended by an appeal to greater security and not by the (vain) hope of being able to cope with greater complexity. Computing scientists should be aware of the fact that tools have the unfortunate tendency to backfire. (Easing the correction of errors immediately leads to more errors being made.) Seeing how CAD for chips has deteriorated into the mechanical production of larger and larger multi-coloured posters, I doubt that the industry will exercise the proper restraint once it has recognized the possibility and potential of mechanical verification. But warnings, no matter how likely to remain unheeded, can never harm.

* * *

Back to our original question: can computer science save the computer industry? My answer is "If the computer industry can be saved, only

computing science can do it.". But it may take a long time before the computer industry – in particular the well-established companies – will share this view. It will almost certainly take longer than the limited period over which they plan their futures. In the mean time, the academic world – which traditionally plans much further ahead – has no choice. It has to refine and to teach to the best of its abilities how computing should be done; would it ever yield to the pressure to propagate the malpractice of today, it had better fold up.

Nobody needs to point out to me that the unescapable duty to train misfits creates its problems: it does so for the University, for its faculty, and for its students. But no one ever heard me suggest that doing science was a soft proposition. But, the more misfits we train, the shorter the time of transition. And this is a point of consideration: if the current malpractice continues too long, the computer industry will collapse and it is hard to save an industry that is no longer there.

Austin, 1 May 1985

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