Answers to questions from students of Software Engineering.

[The approximate reconstruction of the questions is left as an exercise to the reader.]

• Gadgets are not necessarily an improvement, vide the succession

Blackboard → Overhead Projector → PowerPoint

• And I don't need to waste my time with a computer just because I am a computer scientist.
[Medical researchers are not required to suffer from the diseases they investigate.]

• It is not the business of computing science to promote "computerization", say by developing demanding applications so as to create a market for the next generation of hardware.
[Medical researchers are not required to develop new diseases so as to create a market for more pharmaceutical products.]

• It is not the task of the University to offer what society asks for, but to give what society needs.
[The things society asks for are generally understood, and you don't need a University

for that; the University has to offer what no one else can provide.]

• We are all shaped by the tools we use, in particular: the formalisms we use shape our thinking habits, for better or for worse, and that means that we have to be very careful in the choice of what we learn and teach, for unlearning is not really possible.

[Many years ago, if I could use a new assistant, one prerequisite would be "No prior exposure to FORTRAN", and at high-schools in Siberia, the teaching of BASIC was not allowed.]

• A programmer has to be able to demonstrate that his program has the required properties. If this comes as an afterthought, it is all but certain that he won't be able to meet this obligation: only if he allows this obligation to influence his design, there is hope he can meet it. Pure a posteriori verification denies you that wholesome influence and is therefore putting the cart before the horse, but that is exactly what happens in the software houses where "programming" and "quality assurance" are done by different groups. [Needless to say, those houses deliver without warranty.]

• The required techniques of effective reasoning are pretty formal, but as long as programming is done by people that don't master them, the software crisis will remain with us and will be considered an incurable disease. And you know what incurable diseases do: they invite the quacks and charlatans in, who in this case take the form of Software Engineering Gurus.

• Some of you doubt that aforementioned "techniques of effective reasoning", nice as they are for small programs, will scale up, I quote "given the daunting size and sheer complexity of most programs". Well, they will be powerless if you try to use them to disentangle the horrendous mess produced by a group of incompetent, unorganized programmers. Their power manifests itself in the construction phase where (i) they tend to lead to much shorter texts than would be produced otherwise and (ii) lengths of program derivations tend to grow not much more than linearly with the lengths of the programs derived. Finally the programs thus produced are infinitely better than the usual junk.

We should never forget that programmers

live in a world of artefacts, a fact that distinguishes them from most other scientists. The programmer should not ask how applicable the techniques of sound programming are, he should create a world in which they are applicable: it is his <u>only</u> way of delivering a high-quality design. To which I should add a quotation from EWD898 (1984)

> "Machine capacities now give us room galore for making a mess of it. Opportunities unlimited for fouling things up! Developing the austere intellectual discipline of keeping things sufficiently simple is in this environment a formidable challenge, both technically and educationally."

• In reply to questions why we teach useless things that industry ignores, I refer you to EWD920 (1985). Let me quote here one paragraph

> "Back to our original question: can computing 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 aca-

demic 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.".

But to stress how much patience we need, let me give you another old quotation (from 1988)

"Too few people recognize that the high technology so celebrated today is essentially a mathematical technology."

(from the 2nd David-report, so named after the committee's chairman Dr. E.E. David Jr.)

• No, I'm afraid that Computing Science has suffered from the popularity of the Internet. It has attracted an increasing —not to say: overwhelming!— number of students with very little scientific inclination and in research it has only strengthened the prevailing (and somewhat vulgar) obsession with speed and capacity.

• Yes, I share your concern: how to program well —though a teachable topic— is hardly taught. The situation is similar to that in mathematics, where the explicit curriculum is confined to mathematical results; how to do mathematics is something the student must absorb by osmosis, so to speak. One reason for preferring symbol-manipulating, calculating arguments is that their design is much better teachable than the design of verbal/pictorial arguments. Large-scale introduction of courses on such calculational methodology, however, would encounter unsurmountable political problems.

• In the software business there are many enterprises for which it is not clear that science can help them; that science should try is not clear either.

Austin, 28 November 2000

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