By way of introduction

Because most of this introduction is non-technical and yet I want you to believe me, I feel compelled to offer you my credentials. I have now been employed for 37 years, 30% at a Government-sponsored Research Institute, 30% at a computer manufacturer, and 40% at Universities, and most of this at both sides of the Atlantic Ocean. I want you to be convinced that I know what I am talking about.

If we want to understand the world around us, we must be willing to face the facts, even facts we usually don't talk about because we have been educated to be a little bit ashamed of them. Here is one such fact:

> Science is hated because its mastery requires too much hard work, and, by the same token, its practitioners, the scientists, are hated because of the power they derive from it.

Let me repeat this, so as to help it to sink in:

> Science is hated because its mastery requires too much hard work, and, by the same token, its practitioners, the scientists, are hated because of the power they derive from it.

I should repeat it a third time because whatever I say three times is true. Deem that done.

From the above, please don't conclude that I am bitter, for I am not; I am just realistic: destroying scientists and their work is a time-honoured tradition. Archimedes was murdered by the Romans, Hypatia was murdered by the Christians, the Moslims burned down the library of Alexandria, and in 1794, Lavoisier lost his head on the guillotine because of the verdict of the populace that the French Revolution had no use for scientists. (Less than a decade later, France had its next dictator, but the rabble never made the connection.)

For reasons I don't quite understand - partly, probably lack of instrumentation - the science of chemistry was very slow to emerge. So slow, as a matter of fact, that in the first decade of this century, Albert Einstein developed the theory of Brownian motion because of the large number of chemists that still refused to believe in the atomicity of matter. Because Brownian motion can be seen through the microscope and "seeing is believing", Einstein naively thought that an atomistic explanation of the quantitative, observable aspects of the phenomenon would be convincing. This was very naive - but don't blame Einstein, who was very young - for he only convinced the already converted; the stubborn chemists found it much easier to declare Einstein's theory of Brownian

motion to be "too mathematical".

When the science of chemistry eventually emerged, alchemy had already all but disappeared. Chemically speaking, there has been an intellectual interregnum, but I invite you to join me for a short while in the thought experiment of envisaging chemistry with alchemy still in existence. Needless to say, almost all the funding goes to the alchemists, who, in their effort to make gold from cheap base materials, at least attack a problem of the highest social relevance, viz. enable the government to finance its wars without inflicting poverty on the people. The chemists are derided because none of their work has contributed anything to the central problem of gold-making. When chemistry then makes assumptions about matter that would exclude gold-making, it is accused of being counterproductive, demoralizing, and harmful to the national interest. By the time chemistry accurately predicts the failure of alchemy's next attempt, derision turns into open hostility, and, accused of wasting the taxpayer's money, several state universities are forced to close down their chemistry departments. Well, I think that that is enough for our thought experiment: phantasy has already come too close to reality.

Well, we all know what happened. Chemistry is accepted as a science: we hate it for its pollution and our dependency on its products, but we no longer blame it for not trying to make gold. Medicine has been accepted as a science: we hate it for the overpopulation and the soaring medical bill but no longer blame it for not producing the Elixir that gives eternal youth. Astronomy has been accepted as a science; we hate it for removing the earth from the center of the universe but no longer blame it for not realizing the astrologer's dream of accurate prediction of the future. They are accepted as sciences; at the same time, the flourishing business in Healing Gems and Crystals, the horoscopes in otherwise respectable magazines, and governments relying on astrologers are a healthy reminder that Science as such remains rejected and that the old dreams linger on. Finally, one general remark about how sciences emerge: sciences become respectable by confining themselves to the feasible and successful by allowing themselves to be opportunity-driven rather than mission-oriented. (This, by the way, is why managers hate successful science: because it is not mission-oriented, they cannot manage it.)

Let us now turn our attention to computing science. It is hated, of course, like any other science.

In order to understand the specific distaste it evokes, and along with it all the pressures exerted on it, we should be aware of all the unrealistic dreams it failed to realize, for, unrealistic as they might be, the dreams linger on. The dreams have been so many, that I must restrict myself to the major ones.

Surely, our computers would unscramble all the secret code of all our enemies and guide our missiles with unfailing precision right on their targets. Robots would take over the tedium of production, guaranteeing a positive balance of trade for all nations. Office automation would multiply the productivity of the white-collar worker, information systems would enable management to avoid waste and to make the right strategic decisions, and finally, the giant brains would not only relieve us from the tedium but also from the obligation to think about hard problems and from the painful responsibility to take difficult decisions. In short: computers were tolerated because they promised a well-protected and prosperous paradise for the lazy, the incompetent, and the cowardly.

I cannot repeat often enough that, unrealistic as they are, these dreams continue to be dreamt. Their lure is too strong. For instance, the well-

documented decline of the productivity of the American white-collar worker has convincingly been linked to over-automation of the American office, but the "negative" outcome of this study has been totally ignored, in fact so much so that many people involved in computing cannot even believe it.

So much for the unrealistic dreams. But even the majority of at face-value feasible projects went awry. As said before, sciences become respectable by confining themselves to the feasible, and that is precisely what computing science did: it focussed its attention on the intellectual challenge of how to specify and design reliable digital systems of great sophistication. In doing so, it came to a few unescapable conclusions, all clarifying and inspiring for the computing scientists involved, and each of them unacceptable to the world-at-large, that refused to part from its cherished dreams.

For instance, an early conclusion was that the only effective way of winning the battle against unmastered complexity requires that one prevents the latter from entering the design in the first place, or, in more technical terms: mathematical elegance is not a dispensable luxury but a factor that decides between success and failure.

Mathematical elegance, conceptual simplicity, and their companion, brevity of an unambiguous reference manual, are a conditio sine qua non for any product reliable enough to attain stability.

Needless to say, this sober message is unacceptable. Simplicity requires hard work to be obtained and education for its appreciation, and complexity sells much better. I quickly learned this about twenty years ago when I presented orderly program design techniques for the first time in a foreign country. My audience was the personnel of a software house and, in my innocence, I expected for economic reasons this audience to be highly receptive for techniques for making flawless software. I gave a beautiful lecture, which fell flat on its face. The managers were horrified at the suggestion that flawless software should be delivered since the company derived its stability from the subsequent maintenance contracts. The programmers were horrified too: they derived their intellectual excitement from not quite understanding what they were doing and their professional satisfaction from finding weird bugs they had first introduced in their daring irresponsibility. Teaching the competence to program boils down to the training of misfits.

A more technical finding of computing science was that the reliable design of sophisticated digital systems requires much more formal manipulation than any design task faced before. Moreover, the job requires a degree of formalization in comparison to which classical mathematical reasoning becomes slipshod handwaving. Unavoidably, programming is becoming applied logic, but this time applied on a scale that was never envisaged by the logicians. So, computing science found itself faced with the task of figuring out how to apply logic effectively on an unprecedented scale. By now, the competent computing scientist has greatly improved his insight in how to exploit his powers of abstraction for the effective disentanglement of his formal derivations and knows how to design his notations so that his formulae are geared to his manipulative needs and he can let the symbols do the work.

All this is very encouraging and inspiring for the computing scientists involved, but for the world-at-large the message that competent programming is -and unavoidably has to be- VSLAL (= Very Large-Scale Application of Logic) is unpalatable. So what does the world-at-large do? Quite predictably, it tries to ignore the message, and each time the message raises its ugly head it desperately tries to argue that the message

is false or irrelevant. It is really quite pathetic, but any argument, no matter how crooked, that we can do without VLSAL is so welcome that any snake oil merchant in programming methodology finds a ready market for his wares, no matter how ridiculous.

I am not exaggerating. Let me give you a few examples of how expert we have become in sweeping the programming problem under the rug.

Just when the computing science community has agreed that simplicity is an essential ingredient of reliability, the DoD adopts Ada! What more can I say? When the late Andrei Ershov from Novosibirsk asked me my opinion about Ada, I told him that I shuddered at the thought that Western security would depend on it and that I would feel much better if I knew that the Red Army had adopted Ada as well. Andrei smiled and said "Don't worry.".

Our own ACM is also great at skirting the programming problem. Did you ever notice that there is no Special Interest Group on programming? Did you ever notice that the ACM's youngest journal, TOPLAS is not on programming but on Programming Languages and Systems (whatever the latter may be)? Did you notice that the report on the CS curriculum

that was recently published in the CACM carefully spent more space to arguing that the importance of programming should not be overemphasized than to arguing the need to include logic? Such a report, written by many people is, of course, a political compromise, but this compromise was very revealing. Also a bit sickening.

Another revealing phenomenon was the welcome with which the paper was greeted in which Lipton, Perlis and DeMillo tried to argue why proving the correctness of programs was a misguided effort. Thank God that programming had nothing to do with Mathematics! In their eagerness to believe these authors, people failed to notice that the argument was based on the consensus model for informal classical mathematics, i.e. the way of doing mathematics that computing science had already identified as inadequate. Program derivation is not a social process but a formal calculation. The paper was nonsensical.

A very different way of arguing the irrelevance of striving for flawless programs I learned the other month from Jim Gray. Though he addressed an academic audience, he did not give a lecture but a sales talk for "fault tolerance". Well, that need not deter us: their unashamed appeal to our base instincts makes sales talks at least as

interesting as the average lecture. His great theme was that software correctness did not matter at all – and I must confess that this idea seemed very attractive to quite a number of our students. His first argument was a familiar one: the fact that there is at least one known bug for every thousand lines of its software did not bother IBM at all. Now, firstly, this is a lie for the company is very worried by the run-away complexity of its software and the ever more daunting task of more or less "maintaining" that software. Secondly, I can recommend to no one to adopt IBM's quality standards as his own. Gray's second argument was new for me: most software errors were soft. By this he meant that if something did not work as it should, you could either avoid using it or try again, for if the malfunctioning was due to an unproperly resolved race condition, the error may not show up the second time. Here the answer is that the theory of how to design systems without race conditions is well-known and that it is much more efficient -and cheaper!– to apply it than to ignore it. To Gray's advantage I should add that he was quite honest about his position: his only concern was that his systems remained "working", and whether the results they then produced were any good was not his department. As long as your

stuff was better than what an army of COBOL programmers would build on top of it, you were okay. (By the time he made this remark I felt free to leave the lecture hall.)

As final example of skirting the programming issue I mention "software engineering", the IPSE's, the ASPE's, the FRIPSE's and the GRIPSE's and all further animation tools you can think of. They come from a world that has accepted as its charter "How to program if you cannot.". The best of these efforts just confuse composing with the physical act of writing the score, others are foolish at best, and some are criminal.

So much for the world in which we are supposed to educate young people in computing.

* * *

How well did our departments of computing science educate? As I said before, the training of the competence to program boils down to the training of misfits, and under the circumstances sketched above it is only to be expected that our departments of computing science did not educate too well. Regrettably, I feel compelled to put it much stronger: they failed miserably in their educational task by not taking the

intellectual challenge of programming seriously and by not changing the layman's view of programming, but by adopting it instead.

That layman's view was driven home to me almost a decade ago when I gave a one-week course on programming methodology. I had in my audience the manager of software at a huge bank and I knew he was in trouble. After three days he told me that, as far as he was concerned, I was teaching useless stuff, though he fully agreed that if their software had been developed in the way shown, most of his problems would not have emerged. Yet he rejected my recommendations because they would have required him to train his programmers better and he did not dare to do so for fear of losing them to the competition. Today's manager is so horrified at the suggestion of depending on the competence of his programmers that it is psychologically impossible for him to admit that programming should be regarded as a high-technology activity, as an area par excellence for the application of the most effective techniques of scientific thought. His conclusion is that programming does not require any sophistication because it should not, and this is the attitude that is faithfully reflected in many computing

science curricula. Introductory programming courses have, in fact, become a prime target of curriculum infantilization and, instead of teaching the Very Large Scale Application of Logic, they give the student no more than the most primitive operational arguments for thinking about his programs. No wonder that the "Laboratory" in which the student can "test" his programs becomes an essential ingredient of the educational machinery. No wonder that such introductory programming courses — insipid, inadequate, obsolete, and time-consuming — "turn away the best students, who want a greater challenge". [0]

Dealing in our curriculum with programming as a scientific challenge of high calibre is easier said than done: besides the resistance in the rest of the world, you would have to overcome the resistance in your own department.

Firstly, the average computing science faculty member considers the problem of programming decently far below his dignity, while at the same time he has not the foggiest notion of how to do it. Again I am not exaggerating: I

[0] Computing as a Discipline, P.J. Denning et al., CACM, Jan. 1989 (32,1) pp. 9-23

have taken extensive experiments with CS faculty members from all over the world, and the vast majority of them —I mean about 95% of them— cannot program a Binary Search. It is a very shocking percentage, but you can take my word for it. This wide-spread incompetence is, of course, a severe problem for educating students may be hard, but, as we all know, educating faculty members is near to impossible.

A second source of resistance within your department will be formed by those colleagues that have unwisely based their research efforts on the assumed dogma that competent programming is an illusion. Really teaching how to program would dissolve their research topic and the significance of their expertise would evaporate. You cannot expect them to take kindly to that.

Austin, 12 February 1989

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