The nature of my research and why I do it.

(Luncheon speech for Las Cruces, 14 November 1986.)

In 1955, as a young and promising theoretical physicist, I decided to devote my scientific life to programming instead, because I felt that programming well presented the greater intellectual challenge.

The next decade was devoted to gathering experience by designing and implementing systems of increasing size, ambition, and sophistication. By 1965 I could forcefully argue that the ability of detecting in an early stage the need for some special-purpose theory was one of the programmer's most important assets.

Another five years later I formulated the core of the programming challenge as how to prevent the generation of unmanageable complexity. It became clear --to me at least-- that the challenge was a mathematical one. By 1975 I had developed a formal system in which the program and its correctness proof could be designed hand in hand.

I was very excited because that was really great: another area of human endeavour had been shown to be in principle amenable to mathematical treatment.

From a purely technical point of view it is in retrospect amazing that it took me no less than two decades to reach that stage. From a sociological point of view, this slowness is easy to understand: the conclusion was almost universally unpopular. Programmers, and even more so their industrial managers, decried the whole idea as too academic to be of any practical significance. (Many of them still do.) The mathematicians ignored or denied the conclusion, because the type of mathematics needed was so different from what they were used to that they hardly recognized it as mathematics. (Many of them still don't.) At the time I did not understand the vigour of the academic opposition, but it quickly convinced me that computing science had to develop itself the type of mathematics it needed. In retrospect I can understand the rejection: the stress must have seemed all wrong. In a nutshell: whereas during the last century mathematics as intellectual discipline has become more and more knowledge-oriented, most of that knowledge was totally irrelevant for us; for us, for instance, the choice of notation seemed much more often to be the crucial decision.

So I found myself analyzing which habits and which precautions should be cultivated for the mathematical answer to the programming challenge. In the beginning I booked these explorations under the to me familiar heading of "programming methodology".

But then all sorts of things started happening. By virtue of its mechanical interpretability, a programming language inevitably represents a formal system of some sort, and hence program derivation is unavoidably a highly formal activity. But this activity is only doable provided we find ways of preventing the amount of formal labour needed from exploding. As soon as brevity becomes a conscious goal, one begins to ask oneself "What makes formal derivations lengthy?" Not surprisingly, case analysis emerges as a main culprit. So one asks oneself "What techniques do we have for avoiding case analysis?". And one finds a few, some of them sometimes even very effective. (Later in this talk, as time permits, I shall mention some of them.)

Shortly thereafter, the reading of a well-known, generally recommended and widely used mathematical textbook leaves one completely aghast! The arguments --and even the formulations of the theorems-- are often so incredibly clumsy that one is forced to conclude that one's explorations have a bearing on mathematics in general.

Example. In "An Introduction to Number Theory" by Harold M.Stark, a main theorem is formulated as follows:

"If n is an integer greater than 1 , then either n is prime or n is a finite product of primes."

But 1 is certainly finite, and by defining the product of 1 factor --how else?-- to be equal to that factor we can do away with the case distinction in the consequent:

"If n is an integer greater than 1 , then n is a finite product of primes."

But also 0 is certainly finite and by defining the product of 0 factors --how else?-- to be equal to 1 we can do away with the exception:

"If n is a positive integer, then n is a finite product of primes."

Finally we can do away with the identifier n :

"A positive integer is a finite product of primes."

(End of Example.)

In connection with the above example I would like to stress that a clumsy formulation of a theorem is a prototype of an unfortunate interface in the sense that it tends to increase labour on both sides of the fence: it makes the theorem harder to prove and harder to use.

Thus I found myself drawn into a wider topic than just "programming methodology". Sometimes I call it --mainly for private purposes-- "the streamlining of the mathematical argument", sometimes I call it "mathematical methodology", a name partly chosen to express that we were exploring to what extent the experience gained from programming methodology could be carried over to mathematics in general, and partly chosen because it reflected a shift in emphasis, viz. from polishing existing arguments to the discipline of designing in an orderly fashion new arguments. As time went by, heuristics entered the picture.

In these activities we found a new appreciation for formalisms as an alternative to verbal thinking. We learned that formalisms are much more than a shorthand: we now consider then as the main means for freeing ourselves from the shackles of our native tongues.

Let me give you two examples of how confusing our languages are. By the Law of the Counterpositive, $A \Rightarrow B$ is the same as $\neg B \Rightarrow \neg A$. The implication is linguistically rendered by prefixing the antecedent with "if" , e.g.

"It will rain tomorrow if the wind does not turn." ,

a perfectly acceptable sentence. The counterpositive, however, yields:

"The wind turns if it won't rain tomorrow." ,

a funny statement, to say the least. Evidently, the conjunctive "if" carries with it a whole extra-logical burden of before/after or of cause/effect (a dichotomy for which there is no place in the inanimate world). Verbal mathematics has learned to strip the conjunctive "if" from that extra-logical burden, but only to a certain extent, as is shown by the following example. Logical equivalence, alias the mutual implication, is verbally rendered by "if and only if", as in

"An integer n is even if and only if $n+1$ is odd."

So far so good, but have now a look at the following sentence:

"I see with both eyes if and only if I see with one eye if and only if I am blind." .

Linguistically, this is complete gibberish because the sentence can be parsed in two different ways. Its blatant syntactical ambiguity prevents it from being an acceptable sentence. Logically, it is of the form $A \equiv B \equiv C$ and we are, indeed, free to omit the parentheses since the boolean operator of equivalence is associative: the continued equivalence means that the number of false operands is even, and since I do not have more than 2 eyes, the above statement, though baffling gibberish, is formally correct.

In mathematical reasoning, the equivalence is a grossly underexploited relation, and there is no doubt in my mind that this is closely related to the linguistic shortcomings of the verbal rendering "if and only if" . Consequently, I have come to the conclusion that it is a mistake to teach logic by translating formulae into prose, for that is precisely the vehicle from which we want to increase our distance. My characterization "grossly underexploited" is no exaggeration: the judicious use of the equivalence has reduced many an existing argument by an order of magnitude. (I have seen reduction factors of 16 and 24.) The explanation is probably to be found in the nice properties of the equivalence, such as symmetry, associativity and the fact that disjuction distributes over it; moreover it has the full power of equality, i.e. $x = y \Rrightarrow f.x = f.y$.

Formalization, in general the reduction of a greater part of mathematics to calculation, is not our goal. It just happened, and in this connection I would like to stress that our experience is directly opposite to what is commonly held as unshakable dogma, viz. that formal treatments are always long, tedious, laborious, error-prone and what not. On the contrary, a more calculational treatment has invariably led to a crisper argument. My conclusion is that the bad name of formal techniques is undeserved. Our experience is so striking and, at the same time so contrary to what is generally felt that you may have difficulty in believing me. So let me venture an explanation why so many people have such poor experiences with formal mathematics.

(i) Many established notational conventions are not well geared to our manipulative needs, e.g. by being ambiguous or by denying us the combinatorial freedom we need.

(ii) Some calculational styles introduce many steps that are better avoided. When using an associative operator, one may confine oneself to fully parenthesized expressions, explicitly shuffling, when needed, the parentheses by an appeal to

$$(X \bullet Y) \bullet Z \ = \ X \bullet (Y \bullet Z) \quad .$$

Just dropping the superfluous parentheses is much simpler.

(iii) Many people have failed to develop a sufficiently reliable handwriting: they think that formal mathematics is hard while in actual fact they only lack some simple hand-eye coordination.

It was the dream of Leibniz to reduce all of mathematical reasoning to a universally applicable calculus. Not being Leibniz, we are more modest: we won't hesitate to calculate when it comes in handy.

But Leibniz needs to be mentioned in connection with computing science because it could very well be that that discipline gets the task of realizing most of his dream. The reasons are the following:

- The kind of complexity that computing scientists face makes formal techniques indispensable.
- From his experience, the computing scientist is intimately familiar with the idea of manipulating uninterpreted formulae.
- If the need arises, the computing scientist has the technology for mechanization at his disposal.

If ours is the task of realizing the dream of Leibniz, let us see to it that it does not turn into a nightmare!

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