EWD841.html

"There is no 'royal road' to geometry"

It rains all evening; I walked with the dogs with raincoat, boots and my new self-thinking umbrella (which I bought for less than $3.00 at the Shell filling station). But this note is not about tonight or the rain: it is about my spoiled breakfast. This morning's mail brought me a folder of the Savant Institute, "including all the latest works of James Martin". Hence.

My attention was drawn by the announcement of his "Program design which is provably correct - a quantum leap in software". From the blurb I only reproduce the quotation from the author himself:

> "This report describes a mathematically based approach which is really usable because the mathematics are hidden under the cover of a user-friendly set of tools [a set of user-friendly tools?]. It is one of today's most important breakthroughs in the technology of designing systems.
> While it was generally thought that mathematically provable software was a long way in the future, a new technique has now emerged from the work of Margaret Hamilton and Saydean Zeldin which is both highly powerful and practical. The technique has been automated so that provably correct systems can be designed by persons with no knowledge of either mathematics or programming. The software automatically generates logically guaranteed code. Whereas mathematics like that of Dykstra has been applied to only small programs [by the way, how does J.M. know?], Hamilton and Zeldin's technique has been used successfully with highly complex systems. [Etc.]"

(A warning for eager buyers of this marvel: in contrast to my self-thinking umbrella, the book is not cheap, for Savant Institute charges $200.00 per copy.)

I cannot judge the book, for it has not appeared yet. But we can judge the above quotation!

I have still to encounter the first "correct" system -i.e. meeting its specifications- which is not "provably" so, but in view of his intentions and (his perception of) his "target audience" I propose to forgive James Martin his seemingly pleonastic "provably correct". The core of his claim then boils down to his statement that, thanks to automation, "correct systems can be designed by persons with no knowledge of either mathematics or programming".

This is, indeed, a very important breakthrough in the technology of system design that should entirely change the educational practice at our best Universities, where the Departments of Computing Science now waste most of their time teaching their students mathematics and programming. The moral of James Martin's message is that, at the modest price of $200.00 per student, these Departments can be folded up. His claim therefore deserves our full attention.

If the designer provides no bit of information, he has had no choice and can therefore "design" only one -presumably trivial- system. By providing N bits of information he can express his choice between 2^N systems, the coding system employed determining between which 2^N systems he can choose.

Such a coding system has to satisfy the essential requirement that, by allowing N to grow sufficiently, any conceivable system can be designed. The trouble,

of course, is that most conceivable systems —almost all conceivable systems— are utterly nonsensical in one way or another.

Furthermore the coding system must be such that the designer can know which system corresponds to the bit stream he is providing: without that knowledge he would not know how to build up the latter. This means that the coding system must be such that the designer can combine in some way or another "sensical" substreams into a larger "sensical" (sub)stream of bits. (It is here that coding in machine code, but probably also in COBOL or FORTRAN, shows one of its major shortcomings.)

Should the coding system be such that the designer can provide a bit stream to which corresponds no system at all? The answer is yes, human fallibility being what it is. Ideally each "minor modification" —whatever that may be— of an acceptable bitstream should turn it into one that is rejected but the implied redundancy should not be of the trivial nature of duplication or the addition of parity bits, for then the designer would automate the generation of the redundancy. Type checking is a well-known example of successful mechanical exploitation of redundancy. So are all checks of the context-free syntax.

Another form of non-trivial redundancy is together with an algorithm its functional specification. Whether we should include it in the bit stream provided by the system designer is a reasonable question since this form of redundancy is so non-trivial that it requires a theorem prover to check it.

Avoiding it by omission of one of the two is not

a good proposal. Giving the algorithm and ignoring its functional specification is what has led to the software crisis; furthermore the functional specification cannot be mechanically derived from the algorithm. Giving the functional specification and leaving the derivation of an algorithm satisfying it to the machinery – the dream of PROLOG – is no solution either: first of all it requires a (this time mechanized!) theorem prover to generate an algorithm, secondly the functional specification, when satisfyable, can be satisfied by infinitely many algorithms, most of which are highly undesirable for other reasons. Without the designer going into that reasons and their consequences, the machinery would require magic as a built-in feature to make the appropriate choice – this in addition to the magic required for a general mechanical theorem prover! – .

Viable mechanical verification that an algorithm satisfies its functional specification requires that the proof be broken down by the designer to lemmata for which a decision procedure exists: the moral of the story is that in any case – whether the proof is mechanically verified or not – the designer has to design the correctness proof if he wishes to go beyond providing algorithms and hoping for the best.

It is a consoling thought that for all this neither a training in mathematics – to understand what you are doing – nor a training in programming – to prevent the amount of formal labour required from exploding – is necessary. It is an alarming thought that quite a few gullible managers, who, like our James, don't understand what he is talking about, are only too ready to believe in this next promise

of salvation.

In my title I quoted Euclid's alleged rebuttal to King Ptolemy I. Does it help if I translate it into contemporary American:

"There is no such thing as a free lunch." ?

* * *

As any logician knows –since Gödel and Turing– there are such things as provably undecidable questions and provably uncomputable functions. People that don't know that will believe anything. There is the lovely story of the IBM-employee –I shall not divulge his name– whose research proposal mentioned uncomputable functions. When the proposal was discussed he was asked in disbelief "Do you mean to say that there exist functions that our equipment cannot compute?" upon which, tongue in cheek, the scientist answered: "Indeed, in their current form IBM computers cannot compute all functions.", upon which answer his proposal was approved!

In a way, the 30's were more enlightened than the 60's, 70's, and perhaps the 80's. In my very young days, impossibility results could be published and were not completely ignored. With the increased economic significance of technology, such scientific results are now too unwelcome.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

10 October 1982

prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow