Judging "HOS" from a distance.

EWD852.html

> "[...] She even wrote a book."
> "About agapemones?"
> "Yes. And the Higher Wisdom. And Beautiful Thought. That sort of thing. Full of bad syntax."
> "Oh, lord! Yes—that's pretty awful, isn't it? I can't think why fancy religions should have such a ghastly effect on one's grammar."
> "It's a kind of intellectual rot that sets in, I'm afraid.[...]"
>
> Dorothy L. Sayers in "Gaudy Night".

The majority of people would not dare to open their mouth anymore if they knew how much they revealed by doing so. Are apologies dangerous (Disraeli), so are acknowledgements. A paper is not recommended by a profuse acknowledgement to a colleague whom one knows to be too superficial. Having a technical writer write one's article is not a recommendation either.

* * *

"HOS" stands for "Higher Order Software", originally the name of a methodology, now the name of a company. I know the company's product from four publications: [0], which introduced in 1976 HOS as "a methodology for defining software", [1], which appeared after the foundation of HOS, Inc, [2], which oversells

the product, and [3], which for all its brevity is very illuminating. Here is what I reconstructed from these sources.

Margaret Hamilton (now President HOS, Inc.) and Saydean Zeldin (now Vice-President HOS, Inc.) joined The Charles Stark Draper Laboratory in the mid 60's were they got heavily involved in the programming of the Apollo on-board software and the like. Instead of closing their eyes for all the problems they encountered they have studied them and analyzed their origins, asking themselves all the time how these problems could have been circumvented. In the course of the 70's they show symptoms of proposing a remedy.

Their first proposal has very much the form of an applicative programming language with identified intermediate results. The main motivations for their proposal seem to have been the following:

(i) all sorts of manual checks that had been developed – checks against the use of unitialized variables, type checks, and their embellishments – could be mechanized provided the programs were expressed differently

(ii) the unbridled use of the assignment statement all but defeated most restart requirements

(iii) most multiprogramming designs suffered from a premature decision how the total workload should be spread over a number of (quasi-) concurrent processes.

A notation that primarily describes which values depend on which can obviously alleviate these problems.

Of [0] and [1], the first one is the more technical one. (From the limited information I have about the authors, my guess would be that [0] is more Hamilton and [1] is more Zeldin.) Both papers share the characteristic of being so poorly written as to cast severe doubts on the effectiveness of the refereeing for the journals in question (IEEE Transactions on Software Engineering and The Journal of Systems and Software respectively). It is, in fact, often hard to figure out what the papers are about.

Certainly [0], the earlier one of the two papers, has done its fair share of sowing the seed of confusion. Its title mentions "defining software", but the paper nowhere states what is meant by that expression. Here [3] gives the solution: they meant describing an algorithm in a way fit for mechanical execution, but that is what is usually called "programming". A nice example of gobbledegook is the following sentence from its abstract: "We envision a scheme in which the definition of a given system can be described with an HOS specification language which, by its very nature, enforces the axioms with the use of each construct." Just a few questions about this sentence:

(i) what is meant by "a scheme"?
(ii) how has the "given system" been given?
(iii) if we can "describe a definition", what is the (profound) difference between a definition and its description?
(iv) what means "enforcing an axiom" and how does

one do it?
I know that writing an abstract is a very difficult task, but there is no excuse for a sentence like the one quoted, not even in an abstract. Regrettably, similarly loose sentences occur throughout the paper, e.g. "The Structuring Executive Analyzer is a lower virtual layer module with respect to a given hierarchical HOS system in its dynamic state."

The linguistic quality of [1] is amply illustrated by the following quotation: "A system development process is a system that develops another system. Such a system can be viewed as a process [etc.]". The quotation makes one wonder why we need the different terms "system" and "process" (or whether we need them at all). Once every n sentences (for rather small n) one has not the foggiest notion what the authors are writing about and the reader's self-respect can only be saved by the probably correct assumption that they wrote about nothing. (Though applauding the traditional anonymity of the referee, I regret it this time: it always helps to know whose judgement to ignore.)

A poor presentation of excellent subject matter is in principle perhaps thinkable, but the papers do not provide an instance of that rare (and unlikely) occurrence. The notational conventions adopted in the formal text and the ways in which they are

are used are not convincing; I mention just a few examples.

(i) They write "Factor(GCD(n_1,n_2), n_1) = True" where "Factor(GCD(n_1,n_2), n_1)" would have done; I conclude that the authors have only a partial understanding of Boolean expressions.

(ii) Though they also use "And" as infix operator and seem to allow "$x_0 \neq 0$ And $y_0 \neq 0$ And $z_0 \neq 0$", in AXES "And" is a function name, forcing the user to write "And(x_0, And(y_0,z_0))" or, alternatively, "And(And(x_0,y_0), z_0)". For associative operators the infix notation is, however, definitely preferable.

(iii) When they write "x And y" the "keyword" And is syntactically indistinguishable from an arbitrary identifier.

(iv) Whether to use "or" or "coor" depends on the fact whether selection criterion and subfunction depend on all arguments of the parent function. Independently of the relevance of the distinction, the significance of the redundance should be challenged: the same mechanism that checks their correctness if the text is required to distinguish between or and coor could replace, if necessary, or by coor if the text were allowed always to use or.

(v) The conventions require the introduction of very many identifiers. What is the advantage of writing "y=f(z) z=g(x)" instead of "y=f(g(x))"?

(vi) The lavish use of subscripts reveals a lack of sound mathematical taste.

The overwhelming impression is that the authors have had more experience than education. The burning question then becomes "What kind of experience?". They have certainly been able to improve upon what they refer to as "conventional techniques", but in some of their examples these are so backward (if not downright stupid) that their achievement becomes rather unconvincing. I quote -from [1]-: "we discovered that many interface errors took place in the implementation phase when programmers would use instructions in an unstructured language, such as "GOTO +3." Errors would creep in when someone would come along, often the same programmer, and inadvertently insert a card between the GOTO instruction and the location at which it should have gone. Once we discovered the amount of errors that resulted, we enforced by standardization the use of instructions such as "GOTO A" rather than "GOTO +3". As a result, such errors never happened again." I can believe that, but my problem is that the stupidity of admitting instructions such as "GOTO +3" was well-identified as such more than 25 years ago!

Similarly, while really professional software development had stopped using flowcharts, say, 20 years ago, both [0] and [1] proudly mention the development of ..... a flowcharter! (The use of AXES, not mentioned in [0], strongly suggests that the latter flowcharter was developed in the second half of the 70's!) It makes one seriously wonder how many of

their development "tools" are, in fact, mechanized aids for the application of obsolete techniques. (I vividly remember an incident in 1969 when an admittedly professionally somewhat old-fashioned professor of computing science lost almost all his credentials with his colleagues — they were, in fact, flabberghasted — when it transpired that he still used flowcharts in his lectures.)

* * *

Of the publications studied, [2] is the most bulky one — as a matter of fact more than 400 pages —. It has about as much technical content as an average tract of Jehova's Witnesses. The book transmits one message very clearly, viz. that for James Martin writing consists of stringing words together he does not know the meaning of; in all other respects the book only adds to the confusion. I quote by way of illustration — [2], p.40 —

"The approach described in this report is different. Instead of applying ad hoc programming constructs, it applies only constructs which are built with mathematical axioms and proofs of correctness. A library of provably correct operations is built. The operations manipulate precisely defined data types by means of provably correct control structures. [...] Most important, the mathematics is hidden from the typical user so that the method is easy to use."

and — [2], p.58 —

"The functions, thus, decompose into primitive functions which are mathematically rigorous. The means of decomposition itself is proven to be correct in terms of the three primitive control structures which are mathematically rigorous. The data types used by the functions are mathematically rigorous. From these three mathematical bases, great cathedrals of complex logic can be built."

Please don't try to understand the above, for that cannot be done. Martin wisely nowhere states what is meant by the magic formula "provably correct". He only states what it does not mean:
– [2], p. 3 –

"When we say "provably correct code" we do not mean that the program is necessarily without fault. [...]. However the majority of bugs in programs today are caused by the mechanics of programming, inconsistent data, sequence errors and so on."

Applicative programs can be translated into imperative ones, and this can be done mechanically. Unavoidably, the resulting imperative programs display a great number of desirable characteristics: they are syntactically correct – it is impossible to construct a correct translator that can produce a syntactically incorrect program –; all variables introduced by the translator are initialized by the program generated – no correct translator introduces

uninitialized variables into the target program – ; etc. In James Martin's parlance "provably correct" means probably no more that by virtue of their mechanical production texts can enjoy certain characteristics.

From a professional point of view, [2] is eminently ignorable. In other respects it is interesting. It is revealing about HOS Inc., about James Martin and about the computing community in general. The company has "perfected" its product, but not by cleaning up the conceptual mess with which it started – [0] talks about "violation of this theorem": how does one violate a theorem? – ; instead it developed ..... a graphics editor! And that is precisely the kind of would-be embellishment the professional software designer is only too happy to ignore. I am afraid that HOS Inc. has done some "market research" and has concluded that the incompetent form a bigger market. I have never had reasons to consider James Martin as a competent computing scientist, but that he is a competent salesman I don't doubt: he must have seen a market for [2] at $200.- a piece. The book is so terrible that that is a depressing thought.

[0] Hamilton, Margaret and Zeldin, Saydean, "Higher Order Software – A Methodology for Defining Software.", IEEE Trans. on Software Engineering Vol. SE-2, Nr.1 (March 1976), 9–32.

[1] Hamilton, M. and Zeldin, S. "The Relationship Between Design and Verification.", Journal of Systems and Software, Vol.1, Nr.1 (1979), 29-56.

[2] Martin, James, "Program Design which Is Provably Correct.", Savant Research Studies, Carnforth, U.K. (1982)

[3] Cushing, Steve, letter to ACM Forum, Comm. ACM, Vol.25, Nr.12 (Dec. 1982), 951.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

7 April 1983

prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow.

P.S. The day after the above was completed I learned that James Martin is on the board of Higher Order Software, Inc..

EWD