THE DANGERS OF COMPUTER-SCIENCE THEORY[1]

DONALD E. KNUTH

Stanford University, Stanford, Calif., USA

The text of my sermon today is taken from Plato (the *Republic*, vii) who said, "I have hardly ever known a mathematician who was able to reason". If we make an unbiased examination of the accomplishments made by mathematicians to the real world of computer programming, we are forced to conclude that, so far, the theory has actually done more harm than good. There are numerous instances in which theoretical 'advances' have actually stifled the development of computer programming, and in some cases they have even made it take several steps backward! I shall discuss a few such cases in this talk.

Last week at the IFIP 71 Congress in Ljubljana, I presented a lecture which had quite a different flavor: I spoke pro-Computer-Science, and I extolled the virtues of the associated mathematical theory. Today, however, I must consider the Methodology and Philosophy of Computer Science, and so I feel it necessary to right the balance and to give an anti-Computer-Science talk. (I hope that by showing the other side of the coin I will not prejudice the sales of the books I have written.)

Perhaps the most famous example of misdirected theory has occurred in connection with random-number generation. Many of you know that sequences of pseudorandom numbers are often generated by the rule

$$x_{n+1} = (ax_n) \bmod m$$

for some multiplier a and some modulus m. For many years such sequences were used successfully, with multipliers a chosen nearly at random. The numbers passed empirical tests for randomness, but no theoretical reason for this was known except that number theory was able to predict that the

[1] The preparation of this paper was supported in part by the National Science Foundation and the National Research Council. My wife and I wish to thank our Rumanian hosts for their extraordinary hospitality.

sequence has a very long period before it begins to repeat. Finally, the first theoretical advance was made: It was proved (cf. GREENBERGER, 1961) that the serial correlation between adjacent members of the sequence, averaged over the entire period, is bounded by

$$|r(x_i, x_{i+1})| < \frac{4}{a} + \frac{16a+28}{m}.$$

As you know, a correlation coefficient of $+1$ or -1 indicates a strong dependency, while a random sequence should have 0 correlation. According to this new theorem, if we choose the multiplier to be

$$a \approx \sqrt{m}$$

we can guarantee a small serial correlation.

As I said, this was the first real theorem about the multiplicative congruential sequences; because of it, people changed the random-number generators they were using, and the result was catastrophic (cf. GREENBERGER, 1965). A multiplier near $\sqrt{m}$ is always bad when *other* tests for randomness are considered. For example, the correlation is indeed nearly zero when averaged over the entire period, but the theory did not take into account the fact that the correlation is nearly $+1$ over half the period and -1 over the other half! It turns out that almost all multipliers are *better* than those near $\sqrt{m}$. Yet the horrible sequence

$$x_{n+1} = (2^{16}+3)x_n \bmod 2^{31}$$

is still being supplied by IBM as the standard random-number generator for use on its System/360 computers.

Such misapplications of theory have been with us since the beginning. From a historical point of view I believe that the very first work on what is now called the theory of computational complexity was the Ph. D. dissertation of DEMUTH (1956), who made a theoretical study of the problem of sorting numbers into order. From a mathematical standpoint, Demuth's thesis was a beautiful piece of work; he defined three classes of automata, and he found reasonably tight bounds on how fast each of these classes of machines is able to sort. But from a practical standpoint, the thesis was of no help.

In fact, one of Demuth's main results was that in a certain sense 'bubble sorting' is the optimum way to sort. During the last three years I have been studying the sorting problem in great detail, and I have therefore analyzed about 30 different sorting methods including the bubble sort.

It turns out that the other 29 methods are always better in practice, in spite of the fact that Demuth had proved the optimality of bubble sorting on a certain peculiar type of machine.

Unfortunately, people still play the same game today with computational complexity theory: Instead of looking for the best way to solve a problem, we first think of an algorithm, and then we look for a sense in which it is optimum!

Traditionally the problem in finding an optimum sorting method is to minimize the number of comparisons between data elements while the sorting takes place. The best method, in the sense that it takes fewer comparisons than any other known scheme, was invented by L. R. FORD and S. M. JOHNSON (1959). But their approach has fortunately never been used by programmers, because the complex method used to decide what comparison to make costs much more time than the comparisons themselves.

The three most commonly known methods of sorting with magnetic tapes are the *balanced merge* (MAUCHLY, 1946), the *cascade merge* (BETZ and CARTER, 1959), and the *polyphase merge* (GILSTAD, 1960). Cascade merging with n tapes is based on an $(n-1)$-way merge, $(n-2)$-way merge, ..., 2-way merge, while polyphase is apparently an improvement since it uses $(n-1)$-way merging throughout. But DAVID FERGUSON (1964, p. 297) noticed that cascade merging is surprisingly better than polyphase on a large number of tapes: Given N records to sort on n tapes, the following asymptotic formulas for the tape read/write time are valid when N and n are large:

$$\text{balanced merge, } N\log N/\log(n/2);$$
$$\text{cascade merge, } N\log N/\log(\pi n/4);$$
$$\text{polyphase merge, } N\log N/\log 4.$$

This is all very fine in theory but almost worthless in practice. In the first place, the number of tapes n must be large; but tapes are very unreliable, I have never seen more than about six tape units simultaneously in working condition! More seriously, these formulas are valid only as N goes to infinity, yet all N records must fit on one finite reel of tape. Commercial tape reels are never more than 2400 feet long, and this means that other factors suppressed in the above formulas actually are more important than the leading terms. For practical sizes of N it turns out therefore that polyphase is superior to cascade, contrary to the formulas.

Incidentally, Professor Karp of Berkeley has proved a beautiful theorem which shows that cascade merging is optimum in the sense that it minimizes the number of phases, over all possible tape merging patterns. But unfortu-

nately this theoretical notion of a 'phase' has no physical significance, it is not the sort of thing anyone would ever want to minimize.

This leads me to quote from Webster's dictionary of the English language (pre-1960), where we find that the verb 'to optimize' means "to view with optimism."

A few months ago I computed the effect of tape rewind time, which the above formulas exclude, and I discovered to my great surprise that the old-fashioned balanced merging scheme was actually better than both polyphase and cascade on six tapes! Thus the theoretical calculations which have been so widely quoted have caused an inferior method to be used.

An overemphasis on asymptotic behavior has often led to similar abuses. For example, some years ago I was preparing part of an operating system where it was necessary to determine whether or not a given record called a 'page' was in the high-speed computer memory. I had just learned the very beautiful method of balanced binary trees devised by the Russian mathematicians ADEL'SON-VEL'SKII and LANDIS (1962), which guarantees that only $O(\log n)$ steps are needed to find a particular page if n pages are present. After I had devised a complicated program using this method, I remembered that the computer I was using had a special search instruction which would do the same job by brute force in $O(n)$ steps. Since this instruction operated at hardware speed, and since the memory size guaranteed that n would never exceed 1000, the brute force method was much faster than the sophisticated $\log n$ method.

Now I should say a few words about *automata theory*. For many years the theory of automata was developing rapidly and solving problems which were ostensibly related to computers; but real programmers could not care less about the automaton theorems because Turing machines were so different from real machines. However, one result was highly touted as the first contribution of automata theory to real programming, an efficient algorithm that was discovered first by the theoreticians, namely, the HENNIE–STEARNS (1966) construction which showed that a k-tape Turing machine can be simulated by a 2-tape machine with only a logarithmic increase in the execution time. This meant for example that sorting could be achieved on two tapes in $O(N(\log N)^2)$ steps, which was much better than the $O(N^2)$ methods known for two tapes. Well, once again the theory did not work in practice; the Hennie–Stearns construction involves writing in the middle of a magnetic tape, which is rather difficult, and it includes a lot of unused blank space on the tape. As I mentioned before, a single tape is only 2400′ long; so the asymptotic formulas do not tell the story. When the Hennie–

Stearns method is actually applied to a tape full of data, almost 40 hours are required, compared with only about 8 hours for the asymptotically slow method.

The theory of automata is slowly changing to a study of random-access computations, and this work promises to be more useful. Last week in Ljubljana, S. A. Cook presented his interesting theorem which states in essence that any algorithm programmable on a certain kind of pushdown automaton can be performed efficiently on a random-access machine, no matter how slowly the pushdown program runs. When I first heard about his theorem last year, I was able to use it to find an efficient pattern-matching procedure; this was the first time in my experience that I had learned something new about real programming by applying automata theory. (Unfortunately I found out that MORRIS, 1970, had independently discovered the same algorithm, a few weeks earlier, without using automata theory at all.)

Is there *any* area (outside of numerical analysis) where mathematical theory has actually helped computer programmers? The theory of languages springs to mind; surely the development of programming languages has been helped by the highly sophisticated theory of mathematical linguistics. But even here the theory has not been an unmixed blessing, and for several years the idea of top-down parsing was unjustly maligned because of misapplied theory. Furthermore, too many problems in mathematical linguistics have been shown to be unsolvable in certain levels of generality, and this has tended to make people afraid to look for solvable subproblems. We tend to forget that every problem we solve is a special case of some recursively unsolvable problem!

Another difficulty with the theory of languages is that it has led to an overemphasis on syntax as opposed to semantics. You all know the old joke about the man who was searching for his lost watch under the lamppost. His friend came up to him and said, "What are you doing?"

"I'm looking for my watch."

"Where did you lose it?"

"Oh, over there, down the street."

"But why are you looking for it here?"

"Because the light is much better here."

For many years there was much light on syntax and very little on semantics; so simple semantic constructions were unnaturally grafted onto syntactic definitions, making rather unwieldy grammars, instead of searching for theories more appropriate to semantics.

Of course you know that the theory of languages has by now become ultrageneralized so that it bears little relation to its practical origins. This is not bad in itself, although sometimes it reminds me of a satirical article published a few years ago by AUSTIN (1967); paraphrasing this article, we should not be surprised to find someday a paper entitled "On triply-degenerate prewaffles having no proper normal subwaffle with the pseudo-A_4 property, dedicated to A. B. Smith on his 19th birthday." Sometimes theories tend to become very baroque! The tendency of modern mathematics to be 'modern' in the sense of 'modern art' has been aptly described in an extraordinary article by HAMMERSLEY (1968) which should be required reading for everyone.

At this point I would like to quote from some lectures on *Pragmatism* (Chapter 2) given by the philosopher William James at the beginning of this century:

> When the first mathematical, logical, and natural uniformities, the first *laws* were discovered, men were so carried away by the clearness, beauty and simplification that resulted, that they believed themselves to have deciphered authentically the eternal thoughts of the Almighty.

You see that computer science has been subject to a recurring problem: Some theory is developed which is very *beautiful*, and too often it is therefore thought to be *relevant*.

An article has recently been published by CHRISTOPHER STRACHEY of Oxford University, entitled "Is Computing Science?" (1970). He presents two tables, one which ranks topics now considered part of computer science in order of their relevance to real programming, and another which ranks those same topics in order of their present state of theoretical development. As you might suspect, the two rankings are in opposite order.

Perhaps this is the way things should be. Maybe theories are more beautiful and more worthy of development if they are further from reality. Some of the examples I have mentioned suggest in fact that it is dangerous even to *try* to develop any theory which is relevant to actual computer programming practice, since the record shows that such theories have usually been misapplied.

Well, I must confess that I have had my tongue in my cheek, in many of the above remarks. When I first prepared this talk, sitting in beautiful Cişmigiu Park, I was not intending to write it down for the published proceedings, and I could not resist the temptation to have some fun giving an unexpected 'methodology' lecture. I have stated the case against computer-science theory as well as I could; but as many of you probably suspect,

I do not really believe everything I said. It is true that theory has often been irrelevant and misapplied; but so what? We get enjoyment and stimulation from abstract theories, and the mental concepts we learn to manipulate as we study them often give us practical insights and skills. On the other hand, practical considerations do not necessarily lead to awkward mathematical problems that are inherently impure or distasteful. In fact I have been spending many years preparing a series of books which attempt to show that there is a great deal of beautiful mathematics which is directly helpful to computer programmers. My experience has been that theories are often more structured and more interesting when they are based on real problems; somehow they are more exciting than completely abstract theories will ever be.

References

Адельсон-Вельский- Т. М., Ландис, Е. М., 1962, *Один алгорифм организации информации*, ДАН СССР, т. 146, стр. 263-266

Austin, A. K., 1967, *Modern research in mathematics*, Mathematics Gazette, vol. 51, pp. 149–150

Betz, B. K. and W. C. Carter, 1959, *New merge sorting techniques*, in: Preprints of Summaries of Papers Presented at the 14th National Meeting, Association for Computing Machinery (Association for Computing Machinery, Cambridge, Mass.)

Demuth, H. B., 1956, *Electronic data sorting*, Ph. D. dissertation, Stanford University

Ferguson, D. E., 1964, *More on merging*, Communications of the Association for Computing Machinery, vol. 7, p. 297

Ford, L. R., Jr. and S. M. Johnson, 1959, *A tournament problem*, American Mathematical Monthly, vol. 66, pp. 387–389

Gilstad, R. L., 1960, *Polyphase merge sorting, an advanced technique*, Proceedings Joint Computer Conference, vol. 18, pp. 143–148

Greenberger, M., 1961, *An a priori determination of serial correlation in computer generated random numbers*, Mathematics of Computation, vol. 15, pp. 383–389

Greenberger, M., 1965, *Method in randomness*, Communications of the Association for Computing Machinery, vol. 8, pp. 177–179

Hammersley, J. M., 1968, *On the enfeeblement of mathematical skills by 'Modern Mathematics' and by similar soft intellectual trash in schools and universities*, Bulletin of the Institute for Mathematics and Its Applications, vol. 4, pp. 66–85

Hennie, F. C. and R. E. Stearns, 1966, *Two-tape simulation of multitape Turing machines*, Journal of the Association for Computing Machinery, vol. 13, pp. 533–546

Mauchly, J. W., 1946, *Sorting and collating*, in: Theory and Techniques for the Design of Electronic Digital Computers, ed. G. W. Patterson, vol. 3, lecture 22

Morris, J. B., Jr. and V. Pratt, 1970, *A linear pattern-matching algorithm*, Technical Report No. 40, University of California, Berkeley, Computation Center

Strachey, C., 1970, *Is computing science?*, Bulletin of the Institute for Mathematics and Its Applications, vol. 6, pp. 80–82