1980

REPLACED

PAGES.

60
23
37

55
-16
38

A. M. Turing Award Lecture

C.A.R. Hoare

27 October 1980

In a historical survey of literature or art, you will often find explanations of the influence of the life of an author or painter upon his works; and such influence is not necessarily regarded as discreditable. However, a mathematician or scientist takes a sterner view; his work must not be influenced by prejudice, emotion, or personal experience; he must proceed objectively from incontrovertible evidence by rigorous reasoning to relevant and general conclusions. Any failure to meet these standards must be suppressed: it can never be reported in the scientific literature.

But that is not how it is, at least not how it has been with me. Both the direction and the content of my scientific work have been directly influenced by my personal experience; and I have learnt as much of value from my experience of failure as from the exercise of pure scientific intellect. What I have learnt by science can be read in my published papers; and its value has been most amply recognised by your generosity in the grant of this A.M. Turing award, for which I am most deeply grateful. I shall show my gratitude in this lecture by talking about my personal experiences, my good fortune, my failures and my lucky escapes. I will tell you what I have learnt from them, in the hope that you too may learn.

So please do not expect this lecture to be a contribution to science, for it disgracefully violates the scientific convention of impersonality. Regard it (perhaps) as a contribution to the history of science, or just as an exercise in literary or artistic biography, and excuse my immodesty that it is self-applied.

What relevance can this have to the theme of this conference, the dawn of the computer age? But all that was in the dim and distant past. What excuse have I for using the opportunity of this Turing Award Lecture to talk of ancient history? My reasons are simple but serious, because I believe that the same mistakes which we made so long ago are being repeated today, on an even grander scale. In April 1975, I was engaged to advise the United States Department of Defence on the standardisation of a new programming language. I recommended that they should select an existing language, and that the most suitable candidates were CORAL 66, already standardised by the British Ministry of Defence, and in use by the armed services; or a technically superior language PASCAL, which had been in successful use for several years. I predicted that if they started immediately on the design of a new language, it would be not much better than existing languages, and might be a lot worse; that it would take at least ten years to reach a state suitable for standardisation; and that there was no one in the United States competent to do the design. My advice was not taken; but my predictions are gradually and inevitably being fulfilled.

I have continued to give the best of my advice to the Department of Defense in the slow evolution of the strawman through the (woodenman, the) tinman to the ironman, and finally the steelman. Much of my detailed advice has been taken, but not my main advice which has been consistent and (increasingly) insistent: make the language simpler; complexity is far too dangerous, its dangers have been discovered in several other languages, and its use in Defense applications could lead to catastrophe.

You must take immediate and drastic measures to simplify the language before it is too late: for with all the force and solemnity that I can muster I declare to you that it is wholly impossible to write one single reliable compiler for a language as complicated as that, — and certainly not the hundreds that will be needed all over the world; and it will be impossible to write a single wholly reliable program in a language as complicated as that, and certainly not the thousands that will be attempted all over the world. And if that does not frighten you, think of the implementors and users in the Soviet Union; do not put such an unreliable language in their hands.

Your safety depends on the reliability of their programs!

(will not be an exploratory space rocket; it will)
to go astray as a result of a programming language
design error. (be an anti-ballistic defense missile,
spreading radiation over your own cities. I hope you
will find out in time that it was (one of yours) not one of theirs.

Do not believe me because of my academic
distinction; if you want that, you should consult
my distinguished predecessors on this rostrum, all well
versed in the art of programming language design,
Kenneth Iverson, Robert Floyd, John Backus, Edsger Dijkstra, Don Knuth,
and many more; I think their advice would be the same as mine.
Do not believe me because of my success in the
past; in the (reliable) implementation of a small language; consult
those who have devoted so much of their intellectual energies
to the (standards implementation of large and unreliable language).
But do believe me because I
have failed, not once but many times; because I
have been the cause and (suffered the) symptoms of failure,
and because I have not been not ashamed on
this occasion of my greatest triumph, to admit it.
And above all, believe yourself: ~~read~~
your are a competent and experienced programmer.
If there is any feature of the language, any
restriction, any side effect, any interaction which
you do not yourself understand, or which you
suspect will make your colleagues programs
difficult to understand, or less reliable.

In December 1968, in a mood of black depression, I attended the critical meeting of the Working Group in Munich. We had before us a longer and more complete (but very obscure) document, full of errors corrected at the last minute, ~~It was still very obscure, but it seemed to describe a language which the Working Group wanted.~~ But By this time some other members of the Group had become disillusioned, both with the language and with its description. In concert we drafted a minority report, expressing in a grave and reserved tone our grave and deeply felt reservations about the whole project. The signatures on this minority report now read as a roll of distinction in Computer Science. But that did not prevent the superior committees in IFIP from suppressing (our minority report). The motto of language design by committee is supplied by Hilaire Belloc.

- But Scientists who ought to know
Assure us that it must be so
- Oh, let us never, never doubt
What nobody is sure about.

I missed the next two meetings of the Working Group in 1967 and 1968; but I continued to receive a series of larger and longer drafts, and I continued to warn my colleagues of the technical flaws and the political dangers of the language. I wrote them a letter in Aug/Sept 1968, criticising the method of description, the inclusion of "features which will create unnecessary difficulties in implementation and use," and "the rate at which changes have been made to the document in the period immediately prior to submission." I expressed my "deep regrets to the whole Working Group, and the authors of the language in particular, that I should have to recommend such negative conclusion to three years of arduous labour. It is a most difficult thing to recognise that a project has not lived up to the high hopes with which it was initiated; but once the recognition has been made, we must exert all our endeavours to prevent wider propagation of our failures and mistakes."

lessons of history are not very encouraging.

It is now the very complexity of the design which poses the gravest dangers for the implementors of the language, the users of the language, and for all of us who have to live or die with the consequences of errors in these programs. For with all the force and solemnity which I can muster I declare to you that there will never be one single wholly reliable compiler for a language of the complexity of ADA — and a hundred such compilers (over the world.) will be required. Furthermore, there will never be a single provably reliable system programmed in a language of the complexity of ADA — and many thousands of such programs will be needed. If that does not frighten you, think of the implementors and users in the Soviet Union. Our safety depends on the reliability of their programs: do not put such an unreliable language in their hands.

```
procedure read_square;
var X, Y: integer;
  begin writeln("Input the square")

    for X := 1 to 4

  begin for Y := 1 to 4
      read(square[X, Y]);
  readln;
  end

end
```

square[

I have been advising this project ever since the strawman was just a little strawbaby. In April 1975 in Washington, I advised that if a single language were to be introduced for all three armed services, then a selection should be made from existing successful languages, like CORAL 66 (used by the British Ministry of Defence) or the technically superior PASCAL. I told them that a design project for a new language would take ten years to reach fruition; and even then the language would not be better than existing languages, and might be a lot worse. Further, there was noone in the United States competent in the area of language design. My predictions are inexorably being fulfilled.

A. M. Turing Award Lecture

C. A. R. Hoare

27 Oct. 1980
Nashville, Tennessee, U.S.A.

My first and most pleasant duty in this lecture is to express my humble gratitude to the Association For Computing Machinery for the great honour which you have bestowed upon me, and for this opportunity to address you on a topic of my choice. And in all humility, the topic which I have chosen is one of enduring interest to me — it is myself. I want to share with you some of my personal experiences — my modest successes, and more important, my failures and lucky escapes, because

I have learnt more from my mistakes than can ever be revealed in the cold print of a scientific article; and this (lecture) is my chance to give you the chance to learn from them too.

I start my story in August 1960, when I joined a small computer manufacturing division of Elliott Brothers (London) Ltd, from which I was to receive my whole education in Computer Science. In the next eight years My first task was to implement a library subroutine for a new (for the new Elliott 803 computer) fast method of internal sorting which had just been invented by Shell. I greatly enjoyed the challenge of maximising efficiency in the simple decimal-addressed machine code of those days.

I showed my completed program to my boss and tutor, Pat Shackleton, who was impressed by the tight coding of the innermost loops. I then said timidly that I thought I had invented a sorting method that would usually run faster than Shell's without taking much extra store. He bet me sixpence that I had not; and although my method seemed very difficult to explain, he agreed that I had won my bet.

I continued for a while writing efficient library subroutines; but after six months I was given a much more important task — that of designing a new advanced high level programming language for the Company's new Computer, the Elliott 503, which was to be sixty times faster than the present computer, the 803. This was a task for which I was even less qualified than those who undertake it today. But by great good fortune there came into my hands a copy of the Report on the International Algorithmic Language ALGOL 60. Of course, the full language was obviously too complicated for our customers. How could they ever understand all those <u>begins</u> and <u>ends</u> when even our sales manager couldn't. So I set out to design a simpler language, and programmed a table-driven lexical analyser for it — the only part of an implementation which I could see how to do.

Around Easter 1961, a course on ALGOL 60 was offered in Brighton, England, with Peter Naur, Edsger Dijkstra, and Peter Landin as tutors. I asked to attend this course with my colleagues in the language project, "Jill Pym".

Not only was permission granted, but the (divisional) technical manager Roger Cook and the sales manager Paul King decided to come along as well. I was not a very attentive student. My previous study of the ALGOL 60 report had shown that there was no restriction in the language against (recursively) calling a procedure from within its own body; I strongly suspected that this would be a great help in programming the sorting method which I had earlier found such difficulty in explaining. Thus I wrote the procedure immodestly named "quicksort", on which my career as a computer scientist was founded. But due credit must be paid to the genius of the designers of ALGOL 60, who included recursion in their language

~~of course, I was very lucky; but much~~ of the credit must go to the genius of the programming language ALGOL 60. Without recursion I would not have been able to program or even explain the algorithm.