

31 March 1981

July Fomm
Hoare parking ①

Robert L. Ashenhurt
Editor-in-Chief
University of Chicago
1101 E. 58th Street
Chicago, IL 60637

Forum:

In C.A.R. Hoare's Turing Lecture there are several insinuations regarding Ada. ~~Ada is compared to the likes of Algol 68 and PL/I.~~ Furthermore, programmers of the future are warned against using Ada in life critical systems. The comparisons are unfair and the warning, if heeded is dangerous.

Both Algol 68 and PL/I took years to implement and are recognized as difficult languages to describe, to learn, and to use.

Ada is unique in that there were several public reviews of the requirements for a language to replace the veritable Tower of Babel in DoD embedded software which unfortunately are today coded primarily in assembly language. Yes, Virginia, assembly languages! A great deal of effort by DoD, industry, and even a few academics went into refining the requirement documents which were named "strawman", "woodenman", "tinman", "ironman", and "stoneman" as they became progressively firmer. Neither Algol 68 nor PL/I nor even Pascal can point to a document which could be called a requirements document. Furthermore, none of these languages satisfies the requirements of "steelman". As a result of a public competition which any of the present day detractors could have responded with proposals, four contractors were selected to develop language designs. These were delivered on time and were again subjected to public review. Two contractors were selected to revise and to complete their designs. They did this on time. These were also subjected to public review and the choice was known as "green" and later as Ada. No other language has been subjected to public input or competition among language designs.

As soon as Ada was chosen, implementations were begun and exist today! I have a listing for one of them. It is only 3/4 inch thick, beautifully commented, and is not a subset.

This ^{spring} ~~month~~, as scheduled, the DoD competition on designs for a programming environment for Ada which include the compiler, editor, loader, kernal operating system, and data base system will result in three competitive designs being scrutinized by yet another public review.

Robert L. Ashenhurt

30 March 1981

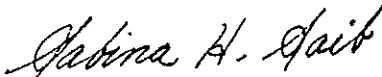
To compare the development of Ada to small languages such as Pascal which is good for teaching, but cannot handle interrupts, low level I/O, or systems programming and which has evolved to so many dialects that portability of programs appears impossible between computers, and to a large language such as Algol 68 which suffers from overcomplexity that cannot be compared to Ada's elegance is far from fair. Yes, folks, Ada is elegant. You may use a subset in the privacy of your own programming shop, but the compiler will provide for all the features so a program written in San Diego will run in New York.

To demonstrate its elegance I close with a version of quicksort written in Ada and in Pascal. While the Pascal version cannot be changed to sort arrays of different types or lengths, it is possible to do so with the Ada version.

Where the DoD and much of industry uses assembly language or at best FORTRAN to program life critical systems, we can hope they will use Ada in the future. Ada has many more mechanisms for catching errors than any of the languages currently used in life critical systems. Implementation was rapid and progress is on schedule.

A consultant to an emperor or to the DoD would be best advised to make a better pair of clothes than advising to discard what is there and to adopt nothing in its place.

Sincerely,



Sabina H. Saib, PhD
Director, Software Quality Dept.

S
H
S
.
.
H
S

Following is a contribution to ACM Forum. Author:

Arthur Evans, Jr.
Bolt, Beranek and Newman Inc.
10 Moulton Street
Cambridge MA 02238

July Form
Hoare package (2)

617-497-3487

C.A.R. 1980
 Professor Hoare's Turing Award Lecture, ~~last October~~, which
~~was~~ reprinted in the February 1981 issue of Communications,
 provided once again a fascinating glimpse into the early days of
 our discipline. It is good that we have the of chance to share
 the experiences of one who was writing useful programs over
 twenty years ago, and I am pleased to have this opportunity to
 express my gratitude to both the Turing Award Committee and to
 Professor Hoare.

After discussing some of what he has learned through his long experience, Hoare chose to attack the design of the language Ada. While I do not claim that Ada is without flaws, or even that his conclusion (that Ada is too big) is necessarily inaccurate, I do feel that the thrust of Hoare's attack seriously misses the point and is not a constructive contribution.

Hoare made clear in his presentation that his professional experience has been restricted to projects of moderate size, projects which are sufficiently small that one person can reasonably comprehend most of the details. While such projects

are surely gratifying to work on, it is unfortunate that problems exist which are bigger than these by several orders of magnitude, problems in which hundreds of programmers labor over a period of years to produce thousands of modules, all of which must work properly together. It was the realization that the tools which can deal more or less successfully with smaller problems are inadequate to deal with larger ones that lead to the design of Ada. Although it will be several years till we can evaluate Ada's success, it is the opinion of many (myself included) that it will meet the needs for which it was designed.

A responsible scientist who chooses to attack the work of others has an obligation to take cognizance of the announced goals of that work and to couch the attack in terms of its failure to meet those goals. Just as it is inappropriate to attack LISP for its weaknesses in solving differential equations (which are not used in AI research), or Pascal for its lack of separate compilation features (which are not needed for classroom exercises), so also an attack on Ada for being too big is inappropriate unless the attack addresses why its size is excessive even for the very large problems for which it is intended. Hoare merely says that Ada is too big and states dogmatically that it ~~won't~~^{will not} work, but he does not say which features he thinks could be removed without compromising its ability to meet its goals. The only language feature explicitly mentioned, exceptions, is too small to contribute appreciably to Ada's size. Further, at least in my opinion, it is a feature which is now well enough understood that it is not "dangerous".

It is surely appropriate for Professor Hoare or any other computer scientist to find fault with Ada, or with any other contribution to our field. Indeed, it is the professional responsibility of every one of us to point out flaws in the language and to make constructive suggestions, since it will be around for a long time and will have a profound influence. (The nature of the sponsor makes this so, regardless of Ada's merits or lack thereof.) However, an attack of the form, "Ada is too big", or "Ada is unsafe", or whatever, is not constructive, does not help computer science, and does not reflect favorably on the attacker, even one with as illustrious a reputation as Professor Hoare's ¹⁹⁸⁰ the Turing lecturers.

Daniel R Hicks
513 Fourth Ave NE Box 815
Byron, MN 55920

July Forum
Hoare package (3)

April 13, 1981

Robert L Ashenhurst
Editor in Chief
Communications of the ACM
University of Chicago
1101 E 58th St
Chicago, IL 60637

Subject: ACM Forum

Dear Mr Ashenhurst

Increasingly over the past several years, articles and letters have been appearing in the professional journals and trade press attempting to explain and solve the "Software Productivity Crisis." ^{i.e.} The Communications of the ACM has ^{i.e.} seemed to have its quota of such articles, and the February issue ^{i.e.} is no exception. ^{of} Communications ^{contains} ~~has several~~ bears on this in several separate places. In his "President's Letter," ^{Peter} Denning argues that conventionally educated programmers are not taught techniques for code saving, sharing, and revising, and, in fact, he makes a convincing argument that the typical college environment actually discourages learning such techniques.

klpl4spl

In the ^{H.E.} ACM Forum, Clarke proposes that the fault lies not with the programmers but with the programs. It is his opinion that a lack of standardization, especially in languages and operating systems, is the cause of poor programmer productivity. He again suggests that we should design one good language to cover all needs. The problem, of course, is to define "one good" language. APL, which he describes as "a very bad language," is actually one of the most standardized and consistent languages in existence. On the other hand, PL/I, the most notable attempt to date to be all things to all people, is frequently decried as obscure and inconsistent.

Over the past few years, I have seen articles which insisted that new languages are needed, articles which insisted that there are already too many languages, and articles (thankfully not recently) which insisted that all programming should be done in ^{assembler} ASSEMBLER language.

I have seen articles touting detailed specifications, code reviews, formal testing cycles, programmer's "workbenches," and proofs of correctness--all in an effort to reduce the cost and improve the reliability of programs. Various authors have argued that colleges should teach more specifics, more general concepts, and an armload of specific curricula.

My own personal experience in college has been reenforced by my contacts with more recent graduates--most colleges (and industry education programs) do a fair-to-good job of teaching programming. Few manage to educate programmers.

Where then lies the problem? In his ¹⁹⁸⁰ ACM Turing Award lecture, ^{also in the February issue,}
^{C.A.R.} Professor Hoare ^{stated,} "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

I can put this in perspective by describing, briefly, my own "education" as an Electrical Engineer. The entire emphasis was on analysis. That is, given a basic circuit design, ^{r/f} analyze the appropriate equations and determine the required values for the various resistors, capacitors, and voltages. Little, if any, emphasis was placed on techniques for the synthesis of a basic circuit design, and never was it pointed out that the basic design could greatly affect both circuit stability and the number of variables which would need to be analyzed.

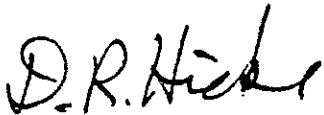
Likewise, most programming curricula emphasize analysis:
Given an algorithm, code an implementation. Given a statement,
describe what it does. Code the same function in three
different languages.

Even structured programming principles are frequently applied
solely as an aid to analysis: If one imposes structure on
a program, it will be easier to analyze.

The problem is that a computer program can be a thousand
times more complex than an electrical circuit, and the first
"basic circuit design" one pulls out of the air is rarely
even close to a good choice. But the analysis-trained
programmer proceeds to do the only thing he has been taught:
analyze the design and, as bugs are found, change a resistor
here and add an IF statement there until all the obvious
bugs have been eliminated or obscured. This results in what
I have called the "First Feasible Solution Syndrome." Any
design is good so long as it cannot be proved to be bad.

The ultimate solution, as I see it, is a basic change to the
way programming (and other engineering disciplines) are
taught. There is little need to dwell on different languages
and algorithms. A good programmer can teach himself a new
language in a few weeks if he needs to know it, and algorithms
will be forgotten next semester anyway. What needs to be

done is to educate (from the Latin "educere"--to draw out) programmers in the management of complexity. Novice programmers especially need to fail--to learn that an 11-line program is not simply one line longer than a 10-line program. They should have the opportunity to construct complex systems so they can learn something about their own limitations. Like ~~Professor~~ Hoare's postmortem of the Mark II project, students should be encouraged to admit where they went wrong. Only this process (abetted by willing students) can produce good designers of software rather than simply programmers. And good software designers can generate reliable code productively with any reasonable language, operating system, debug aids, or building blocks.



D R Hicks

Professor of Computation: C. A. R. Hoare

July Hoare
Hoare package (9) cont

Tel. Oxford 58086

c.c. The Editor
Communications of the ACM

6th May, 1981

Dr. A. Evans Jr.,
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge
Massachusetts 02138
U.S.A.

follows on said text
Dear Dr. Evans,

Thank you for sending me a copy of your contribution to the ACM forum. I think perhaps some of your remarks are based on lack of acquaintance with the relevant facts.

With regard to the letter from Evans I have the following comments:

1. Do you have any evidence that bigger projects require bigger languages? An analogy with any other engineering tool (e.g. pocket calculator) suggests that beyond a certain size, the addition of further complexity is counterproductive, in that it beggars the comprehension of its user. What is needed is to restore simplicity at some higher level of abstraction (e.g. a programmable computer).

2. I am familiar with the goals of the ^{Ada}ADA project, and I have publicly expressed my full agreement with them. My only complaint has been that these goals have been severely compromised by inclusion of too many complex features in the language.

3. The danger of exception handling is that an "exception" is too often a symptom of some entirely unrelated problem. For example, a floating point overflow may be the result of an incorrect pointer use some forty three seconds before; and that was due perhaps to programmer oversight, transient hardware fault, or even a subtle compiler bug. If the programmer uses the first detected symptom (e.g. floating point overflow) as the trigger to fire the rockets, I hope they do not hit anybody important. The right solution is to treat ~~all~~ exceptions in the same way as symptoms of disaster; and switch ~~entire~~ operating (the entire) regime to one designed to survive arbitrary ~~feature~~ entire computer running the program which generated the exception failure of the.

4. A more subtle danger is that the programmer is encouraged to postpone these vitally important considerations of safety, in the hope that he will be able to patch up the problem by exception handlers written afterwards. Experience with PL/1 and MESA ^(show) ~~show~~ that this hope is too often disappointed; and many users of these languages have abandoned the use of exceptions.

5. Exceptions may seem to be a small feature, but their interaction with the other features of the language (particularly tasks) are exceptionally severe. So this feature has a multiplicative rather than additive effect on the complexity of the whole language.

6. I have given my best endeavours over a period of seven years to discharge my scientific responsibilities to the ~~ADA~~^{ADA} project. I have made numerous constructive suggestions to managers of the project, the authors of the strawman to steelman series, and the designers of two of the draft languages; and much of my advice has been taken. All my advice has been placed in the public domain, along with the other project documents, and may be freely consulted.

7. I have given constant encouragement and advice to many competent language designers who are currently engaged in cleaning up and implementing the ~~ADA~~^{Ada} language. ~~I would be glad to help you do so too, if you ask me.~~

8. It is the fate of a consultant to have his sound technical advice ignored for political, commercial, or historical reasons. Very often this does not matter. But when the consultant believes that it matters seriously, and that it is exposing the public to unnecessary risks and delay, then he has a moral duty to give that public a warning, not in a spirit of recrimination, but with the objective of creating a political environment in which vital technical considerations can no longer be ignored.

9. The decision to use the best available public forum for this purpose was not taken lightly, because I knew that in spite of my light hearted tone I should offend many of my colleagues whom I respect for their greater competence and experience in the design and implementation of large scale computer programs. I am sorry that you are among their number, but ^Ithank ^{you} for giving me this chance ^{to put} of putting the record straight.

Yours sincerely,

Arthur Evans is

C.A.R. HOARE
 Professor of Computation
 Oxford University

p.s. You have my permission to publish this letter.

C.A.R. Hoare

July Form
Hoare package (4)

27th April, 1981

Mr. R.L. Ashenurst,
Editor in Chief
University of Chicago
1101 E. 58th Street,
CHICAGO
IL. 60637
U.S.A.

Dear Mr. Ashenurst,

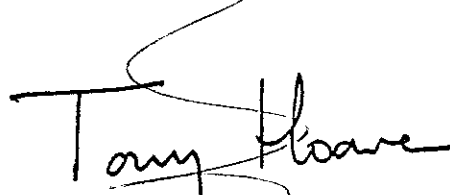
Algol
Ada ~~Thank you for sending me Dr. Saib's letter.~~ I am glad ~~he~~ *Saib* finds
ADA simpler than ~~ALGO~~ 68 and PL/1, but I fear there is little to support
this view. I cannot accept that the extraordinary series of technically
unsound documents entitled "strawman, woodenman, tinman, ironman", and
the technically inadequate management and review processes, ~~and~~ *give* grounds for
confidence in the quality of the resulting design.

Saib The news of the existence of a full implementation of ~~ADA~~ *Ada*
leaves me with mixed feelings. How can that be possible, when the full
language has not yet been fully defined? How could the implementors have
failed to notice this? And - the most important question - how much is
~~Dr. Saib~~ prepared to stake on the correctness of ~~his~~ *The* compiler? The future
of mankind? *cited*

I'm afraid that ~~Dr. Saib~~ *b* is a bit confused. It was the wily
weavers of another story who tricked the Emperor into going forth naked
into the world. The wise tailor of my story merely suggested removal of
some of the stifling layers of outer embroidery which had been overlaid
on top of a basically adequate suit of clothes. And that is what I
specifically recommend for ~~ADA~~ *Ada*.

*continue on
with Evans*

~~Yours sincerely,~~



C.A.R. HOARE
Professor of Computation
Oxford University

July 1981
Hoare package (4)
cont

13 May 1981

continue on after reply to Evans

Dear Mr. Ashenhurst.

Thank you for sending me

Mr. Hick's letter. I think he shows

with regard to the final letter above,
Hick shows

excellent judgement in his analysis
of our problems, both of programming
and of education. The solution he
suggests is the right one — but it is
not easy!

Yours sincerely
Tony Hoare