On a language proposal for the Department of Defense.

"For fools rush in where angels fear to tread."
from "An Essay on Criticism" by Alexander Pope, 1688 - 1744

Introduction.

If some of the following comments sound (unusually) grim, it should be borne in mind that I write these comments not in my native tongue and that combining in English clarity with the subtle use of the understatement is not my greatest strength.

I have screened the report "strawman" -second cut- in my usual way, which I have found most effective in the past, asking myself the following questions.

1) How careful use has been made of the (English) language?
This I have found to be one of the most revealing criteria. It has nothing to do with grammatical pedantry or anything of that sort: I firmly believe that no project can result in a product better than the language in which the product has been discussed and carried out. For anything non-trivial careful use of language is not a luxury, but a must.

2) Is the justification of the goals convincing?
If it is not, it may mean a number of things. It may mean, that the author is concsciously unconvinced of the worthiness of the goals: in that case he should abandon it. It may be, that the author is subconsciously unconvinced of their worthiness, but has been "talked into it": in that case he should clear up his mind.

3) Are the goals compatible with each other and well-understood?
If they are not, early discovery is often possible and then saves a lot of vain effort.

4) Are the most difficult aspects of the project well-identified and is it probable that their difficulties can be surmounted?
No chain is stronger than the weakest link, so search for the latter.

5) Has the author separated his concerns in a helpful manner?
For any complicated undertaking I have learned to appreciate a clear separation of concerns as a (technical) conditio sine qua non; as a result I have learned to appreciate its absence/presence as one of the more telling indications for judging the expected (in)competence for the task at hand.

Whenever on the verge of embarking on a sizeable project --before speeding up towards the point-of-no-return, so to speak-- I myself have always played with respect to my own projects the role of the doubter, of the advocatus diaboli, trying to kill the project. Embarking on an impossible task and failing is a most time-consuming and expensive way to demonstrate its impossibility. What is worse, the failure often fails to convince, because the stubborn optimists will say: "Well, if a next time we avoid those and those mistakes, we may succeed.". In such circumstances a piece of a priori reasoning that argues why a project, even without mistakes in its development, is bound to fail, is, althoug in itself non-constructive, a rewarding exercise. The project "strawman" seems to be in a stage that justifies such a screening; or is its impetus already so large, that the point-of-no-return has been passed?

1 The Use of language.

The spellings --they occur with great "consistancy"-- "kernal" and "encapsolation" are somewhat piculier. But it is not only spelling: in section 2 "A Common Programming Language" (page 5) the author writes about "improvements in ... risks", and he titles a section "Criteria to Aid Existing Software Problems He writes --"IV. A. Criteria to Satisfy Specialized Application Requirements"--

> "... it must be possible to write programs which will continue to operate in the presence of faults."

To start with: "it must be possible" as very ambiguous: either this sentence concludes the existence of a possibility, or states the extreme desirability of a possibility, and by using "must" in this way, the author is close to confusing ends and means. My next objection is that programs don't "operate": a program is as static as a drawing, a proof, a novel or a bill. But, finally, what is meant by "continue to operate"? In the way that we can ensure that a machine will continue to operate by disabling the parity check? I will return to this section later.

On the same page is written

> "Applications which must interface with equipment or people in real time must have access to a real time programming capability."

How can an application interface? How does an application have access to a programming capability? (Reference to "capabilities" in connection with programmin is, I am afraid, nearly always a symptom of fuzzy thinking; I, at least, don't have them for understanding them!)

I stop my linguistic comments here (but not for lack of further material). Pointing out that the signalled linguistic sloppiness is quite usual and accepted in computing circles, is no excuse; for, if it is true, that could very well be an explanation for today's sorry state of the art.

2 Justification of the goals.

A considerable part of the given justification is financial. Now, although coins are very concrete objects, money is a highly abstract notion, and the author does not seem to understand it. How, otherwise, can he write "Software is very expensive"? (III The Most Pressing Software Problems. A.High Cost.) As that paragraph continues, we must conclude that the author would agree to the remark that "poetry and platinum are very inexpensive", because only little money is spent procuring them. As most of the costs involved in the development etc. of software are personel costs

1) the "high price" is due to the great number of programmers engaged; in these times of unemployment, that may not be a bad thing at all
2) alternatively the DoD could halve all programmers' salaries.

I came to the conclusion that money is not the real issue. (I know that those. whose unit of thought is the dollar, will not understand this, and will therefore feel entitled to disagree, but never mind that.)

At various places --and that is a much more forceful justification-- the author expresses his concern about the quality of today's software ("Programming errors can have catastrophic consequences.") and in the section "Timeliness" he justly points out "a common reaction is to aggravate the situation by adding additional manpower". The word "aggravate" is a true one: it is not so much the additional costs involved in adding the additional manpower, it is its degrading

effect upon the quality of the final product that should worry us.

So I decided to read the document as a proposal aimed at improving the quality of DoD software. If this can be achieved, it can probably only be achieved by making fewer mistakes and from that a cost reduction is only to be expected. (Not software, but mistakes are expensive!) In short: I regard a possible cost reduction as a (possibly considerable) fringe benefit, the higher quality as the main goal. I though that I could do so while doing full justice to the author.

3. Are the goals compatible and well-understood?

The author writes "The distinction between high order and low level languages is that between people and machines". I can enlarge on that: in the past, when we used "low level languages" it was considered to be the purpose of our programs to instruct our machines; now, when using "high order languages" we would like to regard it as the purpose of our machines to execute our programs. Run time inefficiency can be viewed as a mismatch between the program as stated and the machinery executing it. The difference between past and present is that in the past the programmer was always blamed for such a mismatch he should have written a more efficient, more "cunning" program! With the programming discipline acquiring some maturity, with a better understanding of what it means to write a program so that the belief in its correctness can be justified, we tend to accept such a program as "a good program" if matching hardware is thinkable, and if with respect to a given machine aforementioned mismatch then occures, we now tend to blame that computer as ill-designed, inadequate and unsuitable for proper usage. In such a situation there are only a few true ways out of the dilemma

1) accept the mismatch
2) continue bit pushing in the old way, with all the known ill effects
3) reject the hardware, because it has been identified as inadequate.

The author concludes his discussion of High Order vs. Low Level Programming Languages with:

> "The fourth approach, and the one which to our mind is most reasonable when object efficiency is important, acknowledges that the major limitation on automated optimization is lack of information. It may have been a mistake to attempt to use programming languages in which the programmer can underspecify his task and hide information from the compiler. [...] High level programs should contain a great deal of information of value to the compiler ..." etc.

But I have some questions. With "information of value to the compiler" is probably meant, information that will cause the compiler to produce a more efficient object program; what kind of "helpful" information does the author have in mind? Does the author believe, that the notion "helpful" is machine independent? And is also compiler independent? If so, why? If not, the purpose of the high order programming language is defeated, for, in order to write a good program, the programmer must then not only know the target machine well, but in addition the HOL compiler for it! Has the author any notion of the algorithms that could be used in the compilers, so as to exploit that additional helpful information? Does the author believe that the desired efficiency can be achieved by such "assisted automated optimization"? At this place --where the controversy has to be talked away-- he suggests so. Later he insists on the possibility to include "machine language"!

When mentioning the possibility of inserting machine language, the author does not seem toc enthusiastic --and rightly so: has he yielded to pressure?--: he mentions that it would defeat the purpose of machine independence. I would like to point out that in all probability the mere possibility of inserting machine language --whether used or not!-- will defeat much more!

1) it may be expected to constrain implementation techniques
2) it may be expected to complicate diagnostics which, remember, should be given in source language components
3) it may be expected to impair security: type checking conventions can be circumvented.
4) it may be expected to impair the ability to exclude in multiprogramming situations a priori a large class of ugly time-dependent bugs. Don't forget that in real time applications, scope rules --ruthlessly checked by the compiler-- are a powerful tool for ensuring that one of the sequential processes will not interfere in an uncontrolled fashion with local variables of one of the other sequential processes of the aggregate. The same holds for a number of independent programs executed in an interleaved fashion; in the latter case it is not unlikely that the correct execution of my program depends on you not having inserted sneaky machine code!

I cannot suggest strongly enough each time to select one of the three ways out of the dilemma, and not to mix them. When the second alternative "continue bit pushing in the old way, with all the known ill effects" is chosen, let that be an activity with which the HOL project does not concern itself: if it does, the "ill effects" will propagate through the whole system.

* * *

I agree with the author's remark in "Simplicity vs. Specialization": "Probably the greatest contributor to unnecessary complexity in programs is the use of overly elaborated languages with large numbers of complex features." I would like to point out that a vital word in that sentence is the word "unnecessary": the remark should not seduce us to believing that programming will become easy when we have a simple programming language, because the intrinsic complexity as might follow from the task, will remain. The author has not analyzed what caused the emergence of those baroque tools, nor does he indicate how he proposes to prevent that his "extensible (!) HOL" results in as many such baroque horrors as extensions are made. The answer "Yes, but that is not the HOL, that is only an extension" is not adequate!

I agree with his remark in "Programming Ease vs. Security"; indeed, many people mistake ease of programming with the ease of making undetected errors. His remarks in 3. "Object Efficiency vs. Clarity and Correctness" are unconclusive; so are the ones in the section 4. "Machine Independence vs. Machine Dependence".

* * *

By the time that I read under the heading Fault Tolerant Programs:

> "In many weapon systems and control programs it must be possible to write programs which will continue to operate in the presence of faults, whether in the computer hardware, in input data, in operator procedures, or in other software. Crucial to fault tolerant programs is the ability of the program to specify the action to be taken at all (!) run time exception conditions." ,

I wonder, whether the author has read what he has written. You see, we have to write "programs which will continue to operate". The language should therefore contain as one of the basic primitive building blocks the procedure called

"get_me_out_of_the_mess"; its body consists of a faithful copy of the Philosopher's Stone.

* * *

4. Are the most difficult aspects identified?

I don't think so. The author has a dream about a nice kernel --and a lot of things he writes about that kernel are quite sensible-- but the word "kernel" itself is an OK-word that should make us suspicious. We have already seen that all machine-dependent characteristics --special peripherals, etc.-- are pushed into the extensions, and it is not inconceivable that that can be done. (In a great number of cases this has been done quite successfully, e.g. a bunch of standard library procedures manipulating an incremental plotter.) By the time, however, that (9. Standard Extensions, c) I read:

> "Any selected common language must be capable of interfacing with data base systems; and, because standards are limited and there is ongoing research in this area, the data base interface should be defined as an extension which can grow at the user level without inventing a new language."

I shudder. This might turn out to be a non-trivial task, and, given a mechanism for "language extension", it is not clear to me at all that that mechanism will enable us to cater for any conceivable data base interface. In all probability it won't. And that would imply that the design of the extension mechanism implicitly delineates the class of data base interfaces to which the HOL can be extended so as to fit. In this sense we can regard the possibilities and limitations of the extension mechanism as a "downward meta-interface" towards the data base. Nowhere the author gives a hint that, for instance, here the HOL-designers have a very serious responsibility. A more careful scrutiny of the proposal, I am afraid, will reveal similar instances of unawareness.

5. Separation of concerns.

The author makes, for instance, an insufficiently clear separation between the definition of the language and its implementation. At "Source Language Characteristics" 3. Variables, Literals, Constants, c (page 10) 14 lines from below, he writes:

> "Whether a variables will be assigned a value is in general unsolvable at compile time, but in those cases in which it is not easily determined by the translator, it will not be easily determined by the programmer and those who must maintain the program and should therefore be considered an error."

I am all in favour of a programming language that does away with the phenomenon of the possibly uninitialized variable, and I fully agree with the author's remark --in the same paragraph-- that automatic initialization by default is a fake solution. The only solution that I know of introduces the initialization syntactically distinguishable from the assignment to a variable that has already a value and ensures by syntactic means that no unitialized variable can ever be accessed; in this case the syntactic means are provided by a refinement of the scope rules. (I once designed such a system for a language with a pronounced sequencing discipline; needless to say, the inclusion of go to's --another of the requirements in the proposal!-- makes this problem very much harder if not insolvable.) Under no circumstances, however, the question whether something is an error or not, should be made dependant on the cleverness of a compiler! And this is exactly what the above, quoted paragraph suggests to me.

6 Additional remarks.

The section 5 "Scope", c. suggests that the scope of a variable declared at the beginning of a block automatically extends over inner blocks --unless a new declaration for that identifier is given for the inner block-- like it was done in ALGOL 60. Whether this automatic inheritance by an inner block of the global variables is a very safe convention is subject to serious doubts. I have experimented with a language in which at the beginning of an inner block all global identifiers identifying variables or constants needed in the inner block had to be listed explicitly --it even indicated the nature of the inheritance: as a constant, as an initialized variable or as a variable that had to be initialized--. I found this form of redundancy very helpful and can even argue why such redundancy is in a sense indispensable.

* * *

The most revealing (and also most alarming) passages, however, I found in the recommendations. "The design of a simple, uniform language ... should be do-able in six months using qualified language designers/implementers." I was flabbergasted! (It taught me, again, that the best way to read documents is often reading them backwards! In cauda venenum.....) The problem is that I happen to know a few people that I would consider to be "qualified language designers/implementers", but none of them would regard the design of a simple, uniform language as only a six-month job! The design of something worth having really takes many times longer. If the six-month period mentioned does not already reveal what the author has in mind, the word "using" does it: how much nicer the quoted sentence would have been, if he had written "by" instead....

The whole document gives me the uneasy feeling that here I see a proposal to concoct yet another nil-solution to an unsoluble problem. The worst shock I got on the next pages "Thus the benefits will grow if new efforts adopt the standard whatever it is." (7. my undelining) and "Research in the design of programming languages ..., in programming methodology ... should continue in parallel with the use of a fixed common language" (10).

If the author knows anything about research, he should know that one of the most difficult mental gymnastics is to dissociate oneself from the inadequate experiences and prejudices collected in one's past. Section 6 "..through alteration of extant languages" strongly suggests a conservative effort. And the outcome of that conservative six-month effort should present the framework (constraints?) for research in programming methodology? Now, come on....

After having stated (7) that the benefits will grow if the new efforts adopt the standard "whatever it is" the text continues ominously "The ultimate measure of success must be in cost-savings and failure of a project to adopt the common language should only be for demonstrateable economic reasons." I don't like that sentence at all. The use of the word "failure" in incriminating in this context, "refusal to adopt the common language" would have made the sentence less offensive. It is perhaps as well that it has been written as it has been: it betrays a dangerous attitude which otherwise could have remained hidden.

Is it not amazing that the lack of "management visibility" can become so frustrating for the managers that people are seriously proposing to enforce a standard "whatever it is"? Such an enforcement could easily stiffle all progress. It has already been suggested the Red China could win the next war provided that it manages to skip the FORTRAN/COBOL stage, which now paralyses much of computing

in the Western hemisphere.

Note. COBOL has been designed with the intention to eliminate programmers. And what is the result? COBOL is only used by professional programmers (by about 75 percent of them, as a matter of fact). As COBOL has been pushed very strongly by the DoD, this failure of COBOL to meet one of its goals must have been noticed within the DoD, and I expect that some experts of the DoD must have investigated why COBOL has failed so miserably in this respect. Has that same group of experts also studied "strawman"? (End of note.)

The burning question that should be considered is, whether the adoption of a common language HOL will really improve "management visibility". I doubt that it will, I seriously doubt.....

Plataanstraat 5
NL-4565 NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow