A letter to Professor Zohar Manna, 26 July 1976.

Dear Zohar,

The fact that this letter has an EWD-number indicates that I intend to give it a somewhat wider distribution than just you. If you wish to answer to it, I can distribute your answer along the same channels, but, of course, only if you so desire and authorize me to do so.

Last week I spoke to one of the attendants of the recent Summer School organized by the Mathematical Centre in Amsterdam, at which you were one of the lecturers. As far as he was concerned you have stolen the show: he was most impressed by your performances and I thought that you might like to hear so. But that is not the reason for writing this letter to you.

This letter has been prompted by the "Stanford Artificial Intelligence Laboratory Memo AIM-281/Computer Science Department Report No. STAN-CS-76-558" of June 1976 and titled:

> Is "sometime" sometimes better than "always"? Intermittent assertions in proving program correctness."

written by you and Richard Waldinger.

One seemingly irrelevant remark first. I hate computer-produced manuscripts with all those different type founts --and all lines right-adjusted!-- if the price to be paid is that all individual letters are ill-shaped. I just cannot read a report like yours without constantly suppressing the desire to clean my glasses because all those fuzzy boundaries are so annoying. I much prefer a well-chosen, single type fount manuscript produced on an excellent typewriter --this, of course, excludes most of the IBM pingpong balls-- and I don't care about the right-adjustment either. (On the contrary: without it, significance can be given to additional spaces.) I consider this would-be service to the reader --or is it a service to the writer?-- as a misuse of electronics that I abhor just as vehemently as soft music in department stores, waiting rooms, airports etc. (The only thing printed decently on your report is the Stanford Seal in red on the cover!) I hope that --possibly after some thinking-- you will agree that this remark --although not so much addressed to you personally-- is not so irrelevant: as computer scientist we have, more than anybody else, I think, a responsibility in trying to prevent computer usage from

degrading our lives. The rest of this letter is addressed to you personally.

You and Richard Waldinger deserve each reader's compliments for the way in which your report has been phrased: for that very reason it was a pleasure and a privilige to read it. I think that I understood every sentence of it, something that is in sharp contrast to what is usually dropped in my mailbox: reports of which I am sure that I cannot understand --nor anybody else for that matter-- at least one out of every eight sentences. Your report is exceptionally well-written. Thank you, it is nice to see indispensable standards sometimes maintained! So much for my compliments, you yourself can undoubtedly think of other nice things to say about it.

When I read the abstract in which you announced to use "assertions that must be true at some time when control is passing through the corresponding point, but that need not be true every time" I was amazed to the point of being intrigued. I tried to think of an example where I would like to do that and could not think of any, so I continued to read the report itself. Afterwards I realized why I could not think of an example: you and I do not mean the same with "control passing through a point".

Let me reproduce, for the sake of clarity of this letter, one of your programs:

```
        input(x  y)
start:
more:   if x = y
        then finish:  output(y)
        else reducex: if x > y
                      then x:= x - y
                           goto reducex
             reducey: if x < y
                      then y:= y - x
                           goto reducey
             goto more.
```

Note. In order to display the symmetry more clearly, I would always have written on line 7

"reducey: if y > x" (End of note.)

Question. Would it not have been more consistent to position the last "goto more"

with the "goto" under the "then" two lines higher? (End of question.)

And in your treatment a vital role is played by Lemma 1:

if sometime $x = a_1$, $y = b_1$ and $x, y > 0$ at more
then sometime $x = y$ and $y = \max\{u: u|a_1 \text{ and } u|b_1\}$ at more

(where "p|q" should be read as "p divides q").

Now my privately preferred representation for that code --with labels inserted for the purpose of this discussion-- is:

```
    x, y := X, Y;
L0: do x > y → L1: x:= x - y
     [] y > x → L2: y:= y - x
    od;
L3: print(y)
```

with at L0 relation P0: $\gcd(x, y) = \gcd(X, Y)$ and $x > 0$ and $y > 0$
with at L1: P0 and $x > y$
with at L2: P0 and $y > x$
with at L3 relation R: $y = \gcd(X, Y)$ because (P0 and not ($x > y$ or $y > x$)) => R .

The reason that I was so amazed by your announcement is that in my view control passes only once through L0 --viz. at the initiation of the do...od-- and only once through L3 --viz. at its completetion-- . If we wish to take a closer look --i.e. inside the repetitive construct-- we may observe control passing a number of times through L1 and L2 .

Comparing my approach to yours I realized that I do not even consider control passing through the guards $x > y$ and $y > x$. I consider the evaluation of the guards $x > y$ and $y > x$ --which as far as I am concerned could take place in parallel-- not as something "through which control passes". In my view the function of the control is to control the execution of the assignment statements; the guards --which in a sense are "part" of the control-- could be evaluated by a separate "control computer", I hardly consider their evaluation as part of the computation proper. The function of the computation proper is to decrease x by y or y by x , "control" has the secondary function to schedule these happenings in such a way that the invariance of P0 is maintained.

In this terminology the observation that "assertions sometimes hold"

boils down to the observation that evaluation of a guard needs only be requested when its value is not known a priori. In that formulation the remark is nearly trivial.

Von Neumann's idea that the processor A that is doing the real work -- i.e. carrying out the assignment statements-- and processor B that is evaluating the guards --i.e. figuring out what A has to do when-- could profitably be merged into a single processor is some sort of a pun. (It is not as bad as his pun that a program should be able to modify its own instructions, but it is still a pun: it does not distinguish between the traffic and the lights!) But this pun is the kind of "flattening" that should only be considered in the specific realm of thought where it belongs, i.e. when we consider in more detail how to embed something with a clear conceptual structure in a homogeneous environment. Can you understand that in retrospect I think it a great pity that Bob Floyd formulated his 1967 article in terms of that other Von Neumann relic, the flowchart language? A choice that is responsible for the all too common misunderstanding that the flow analysis that isolates the "loops" is a key component of the inductive assertion method...

In other words, I have the feeling that part of the problem you feel you have solved has less to do with programming as such than with the underlying --but rather arbitrary-- computational model you have chosen. For already quite some time I have the impression that with respect to both the theory and the practice of programming no significant further progress is possible, unless we postulate our semantics in a way that is absolutely independent of any computational model. This may sound as heresy in your ears --I just don't know!-- but that paper by you and Richard Waldinger has confirmed my impression. The problem with computational models is that, by being overspecific, they are bound to lead one astray. This, indeed, might sound as heresy in the ears of someone who writes in his conclusions about "the way programs work", for that wording betrays an undiluted operational attitude. It is not the program that "works"! Agree?

* * *

Finally a few questions. In your conclusion you write: "If the lemmas and the well-founded orderings for the induction are provided by the programmer, to construct the remainder of the proof appears to be fairly mechanical. On the other hand, to find appropriate lemmas and the corresponding orderings is as difficult a task as finding the invariant assertions and well-founded orderings for the conventional ways of establishing correctness and termination." I believe

you. But should not then the conclusion be that, indeed, the raw code is an inadequate starting point for verification, so inadequate that it is, as a matter of fact, silly to try to use it as such? And should not the conclusion be that the programmer has only done a decent job and has only delivered a useful and complete product, when he provides the lemmas, the invariant assertions etc. as well? Should not the conclusion be that the division of labour, as attempted in the sixties, in which the programmer produced just the raw code, has been proven to be an unworkable interface?

I know that some people find it difficult to draw these conclusions because they are afraid of their consequences; usually they phrase their objections in terms of commiseration with "the average programmer", arguing that we cannot increase the burden on his shoulders. But that argument is a fallacy, because that is not what I am proposing. By teaching the programmer to be aware of the role of the invariant and of the nature of his inductive arguments we give him the mental tools that are indispensable for doing his job properly: as soon as he masters their use he will never again program without them!

Some people won't be able to use these tools to their advantage, but are those able to program now? In any case I don't think that we should allow their presence to prevent us from drawing above conclusions, which should have a profound influence on our teaching. I would like to be sure that the teaching of competent programmers does not meet opposition that is inspired by the hope to keep a target of artificial intelligence alive!

Greetings and best wishes! Yours ever,

Edsger.

Plataanstraat 5
NL-4565 NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow