THE EARLY NORDIC SOFTWARE EFFORT

Ingemar Dahlstrand

Formerly at the Department of Computer Science, University of Lund, Sweden;
Paragrafgränden 26, SE-22647 Lund, Sweden

Abstract: Early Nordic cooperation comprised ideas, programs, even computer blueprints and resulted in rapid progress. Regnecentralen contributed importantly in developing Algol 60. Algol for Facit EDB was used at many sites. Further advances were Datasaab's Algol Genius and the Norwegian Simula project. Regrettably, failure of our compiler companies led to software setbacks. We revisit the Algol debates considering later advances like functional programming. For example, provability of programs depends on recursion and absence of side effects. Computer progress forced cooperation between people from different environments. Too little was done consciously to avoid the ensuing cultural clashes, which impeded progress.

Key words: Software, compilers, Algol 60, Nordic historical efforts

1. EARLY NORDIC PROJECTS AND COOPERATION

1.1 Background

When I started working for the SAAB aircraft company in June 1955, the computer BESK at Matematikmaskinnämnden (MMN) had been in operation for one and a half year and was already running three shifts. The SAAB aircraft company, the University of Lund, and Regnecentralen in Copenhagen had started constructing modified BESK copies, based on blueprints generously supplied by MMN. These went into operation within the next two or three years. The move of the original BESK group from

MMN to Facit Electronics in 1956 started a commercial production of BESK copies called Facit EDB, ten of them altogether. This gradually widened our circle of cooperation.

Software was extremely primitive in 1955 limited to a small library of standard procedures. We heard about a formula translating language FORTRAN, but for us that seemed far beyond reach. In 1956, we received, for the first time, access to an assembler for symbolic addressing. Our group at SAAB translated these tools to the machine code of our BESK copy so we had something to start with. Already then, we discussed software with the people at Regnecentralen, but as long as we worked at assembly language level, slight incompatibilities were enough to block actual program exchange.

1.2 The NordSAM conferences

In 1959 the first Nordic computing symposium, NordSAM, had taken place in the marine base town of Karlskrona. For the next 10 or 12 years, these yearly symposia were the important events of Nordic computer world, the place where ideas were hatched and results reported. (Later they became rather too big and too commercial to fill this role). In Karlskrona we first heard of the ongoing work on Algol, centered in Copenhagen; Regnecentralen administered the Algol Bulletin in which the international discussion about Algol 58 took place. With the publishing of Algol 60, writing an Algol compiler had high priority at Facit Electronics, where I was now working, and they placed me in charge of the project.

1.3 Algol implementation

Our goal was essentially to replace machine coding with Algol programming on all of the ten Facit EDBs. Our company sold some of them and ran some of them on a service bureau basis. It meant that the compiler had to be user friendly; a lot of thought went into identifying source code errors and producing readable decimal memory dumps. We assembled a network of contact persons to handle teaching, counseling, and reporting of compiler errors. The compiler had to produce efficient machine code. To achieve this we sacrificed recursion, and we were prepared to give up other features like expressions called by name ("Jensen's device"), switches and Boolean variables, as long as we could get the compiler to work and handle straight, everyday coding fast and well.

We were two persons full time on the project, Sture Laryd and myself. We divided the work into (1) a preprocessor that caught source code errors and built up an identifier dictionary, (2) a formula translator proper and (3) a

postprocessor that linked intermediate code with standard procedures to produce executable code. Laryd took upon himself the formula translator, which was the most difficult part since we had no experience whatever in this area. He completed the work on time, by and by including almost all of the features that had seemed so difficult at the start.

Given our lack of experience, cooperation with other groups was important, indeed crucial, to success. Via Regnecentralen a journal article from the German ALCOR Group came inte our hands and provided a first foundation. Our visits to Copenhagen and discussions with the people there were invaluable. We learned not only how to implement but also how to interpret the sometimes quite difficult Algol 60 Report. In particular, the procedure part gave us headaches. The final Paris conference in January 1960 had accepted contradictory changes to it, so Peter Naur had to straighten out the whole procedure concept and have it accepted by correspondence. Right away people started to note unexpected effects of the new version; for instance, Jensen discovered his famous device at this time [1].

Our first compiler was ready in October 1961, and was gradually improved up to 1964, taking into account new I/O hardware, slight computer modifications to improve procedure handling, and, of course, the experience gained – and errors found! – through operation. And, we gained a lot of experience indeed. Once Facit-Algol was available, use of assembler languages withered in a couple of years, except for maintenance of existing projects and a few large projects where time or space were of overriding importance [2].

1.4 Failure of hardware project

The hardware program of Facit Electronics was put to test in 1960 when competition with U.S. suppliers became a threatening, and finally overwhelming, reality. A number of important deals were lost to the Americans, and our company ceased building general-purpose hardware in 1964. The existing computing centers of Facit Electronics were reorganized into a service bureau Industridata, with the business idea of selling leftover time on the computers of the owner companies. In spite of this inauspicious start, Industridata continued in business for 20 years and made a place for itself both in technical and administrative ADP.

1.5 Algol-Genius at DataSaab

The Algol group at Facit Electronics took part for a couple of years in compiler work for SAAB, which had gone into commercial computer

building around 1960, and was implementing an amalgam of Algol and Cobol, called Algol-Genius. This computer effort went on until 1976 and scored a number of well-earned successes, including supplying the computers for the Swedish taxation and census authority. Eventually, however, this company, too, failed in the face of stiff international competition.

1.6 Hardware failure sets back software

I think it is safe to claim that our software effort was a success. Modern tools were made available at an early stage and certainly contributed to our countries becoming advanced computer users from that time on. Unhappily, much of the momentum gained was lost through the failure of our hardware effort. The entry of powerful American computers with software in the FORTRAN and COBOL tradition in many places swamped the Algol-using groups. Indeed, they dutifully implemented Algol on many computers sold in Scandinavia; but unless these compilers allowed the user to call FORTRAN procedures, they could not compete with FORTRAN.

It is perhaps not obvious today why that should be so. However, in those days software in general and compilers in particular were not portable and thus closely tied to the associated computer. Computer suppliers developed compilers to sell hardware, computer and software being offered as a package. A famous U.S. court decision in 1964 started the slow "unbundling" process. Suppliers dutifully put separate price tags on hardware and software, but the software tag sometimes seemed more of a hint about the possible bargaining range. It was quite some time before separate software houses became big business.

1.7 Use of the Norwegian Algol successors

In the university world (where I have been since 1968, in Lund) we were lucky to have good Algol compilers at Control Data and Univac. Especially the NU-Algol compiler for the Univac 1100 series, written in Norway as part of the SIMULA project, was reliable, efficient and contained a string extension. This made it one of the most practical programming tools I have come across in my computing life. Simula gradually gained ground and has been used at our Computer Science Department for object-oriented programming and real-time at least until 1997 (when I went into retirement). Our pupils sometimes criticized us for using such an old language, but we could not really find anything newer that was better.

2. ALGOL DEBATES REVISITED

What were, then, the points we discussed in the Algol Bulletin and at our meetings? There are three points I would like to mention; they are typing, side effects, and recursion.

2.1 Typing

In pure Algol 60, it is optional whether to specify the types of formal parameters or not. But we cannot compile a procedure without specifications without following all possible calls of the procedure from the main program. If the actual parameters of these calls happen to conflict it may be necessary to compile several instances of the procedure; compiling it separately from the main program of course is not possible at all. A closely related problem was the existence of procedure parameters that were themselves procedures. This made the tracing of procedure calls very difficult. Since Algol 60 allowed integer labels, it might happen that an actual parameter like the number 3 might end up as a label at the end of one chain of calls and as an integer somewhere else. All this seemed overwhelmingly difficult at the time.

We patched these problems by making specifications mandatory and forbidding integer labels altogether. However, a real problem remained, namely the insufficient typing of procedures and functions. Today we know that the type of a procedure is in fact a composite type, including the types of all its parameters and the type of its function value if it has one. This insight forced itself on the FORTRAN people handling very large programs, where separate compilation is a necessary evil. A Fortran 90 main program has to contain a complete specification of any separately compiled procedure that it calls. In Pascal, a formal parameter that is a procedure always has a composite type specification, even when there is no separate compilation.

Making specification of parameters mandatory deprived us of the possibility of writing generic procedures. There is a legitimate need for generic procedures: for instance, one should only have to write one quicksort procedure and then be able to use this for any type for which we define the relational operators. Put in another way, there should be a way of defining and working with a supertype "sortable". Another such supertype would be "arithmetic", allowing us to write generic matrix procedures.

For other reasons, *explicit* specifications are mostly unnecessary. In a functional programming language like Miranda, we can compute the types of all entities in the program at compile time from the operations used and the calling sequences that are inherent in the program structure. The compiler finds all typing errors in this process, so the typing is strict, or

rather, as strict as necessary. Type specifications may be added to ease understanding, but the only place they may actually be needed are for the input files.

All of this we could not possibly have foreseen in 1960. We did what we had to do just to push through those first implementations. What we perhaps did wrong was that we did not question Algol enough; we did not draw a line between design weaknesses in Algol and our own implementation problems. Algol seemed so brilliant it was almost sacrilege in our environment to criticize it. However much we admire our brain products, we must see their faults and get on with progress – or someone else will.

2.2 Side effects

Let us now turn to the side effects issue and begin by having a look at the famous function Sneaky that Mike Woodger presented to the Algol community in December 1960 [3].

```
real procedure Sneaky (z); real z; value z;
begin Sneaky := z + (z - 2)↑2; W := z + 1 end Sneaky;
...
P := Sneaky(k) * W;
```

It was reasonable to assume that W would enter into this expression with its new value k+1, as a side effect of Sneaky. Suppose we changed the order of the factors? Would W * Sneaky(k) return a different result, based on some previous value of W?

There were three ways of tackling this problem. The computer scientists among us wanted to prescribe the exact order of evaluation of primaries. They wanted every well-formed program to have a defined result. The mathematicians threw up their hands in horror at this. If two expressions were mathematically equivalent, they ought to produce the same result. Therefore, they wanted to forbid a function to change global variables at all. The rest of us we ready to leave the case undefined, thus discouraging sneaky coding. The dispute took on a religious fervor and almost split the Algol community in two.

The computer science side won the day, in the sense that they established evaluation from left to right. However, the mathematicians won the public relations battle, one might say. For tens of years afterwards, new programming languages would prohibit "undesirable side effects" – even in command languages that did not contain functions at all.

It is interesting to see that in Miranda side effects can not take place (since there is no way of changing an entity once bound to a value) and this

is one of the features which make it possible to transform and prove Miranda programs [4]. It is also interesting to note how strict one must be to achieve this freedom from side effects; for instance random numbers in Miranda conceptually form an infinite array, the elements of which are unknown at start and computed as needed. So instead of

```
square(random) ≠ random*random
```

which is a paradox, we have

```
square(random[i]) = random[i]*random[i]
```

whereas

```
square(random[i]) ≠ random[i]*random[i+1]
```

Simple and logical!

2.3 Recursion

Recursion, or more precisely whether to implement it, was a third topic discussed intensely. At Facit we decided not to, which probably gave us a faster executing code, which again probably contributed to the rapid spread of Algol among our customers. However, I do admit I did not realize the importance of recursion for a "clean" implementation. (Ours had a persistent bug or two that propagated to SAAB's Algol-Genius compiler.) Nor did I see that applications in linguistics (including compiling) require recursion; nor did I foresee that recursion is another one of the bases for program proving in the sense of Miranda.

2.4 Input/Output

Input/output was not a topic of strife between the Algol groups, since they left it explicitly to the implementer. The lack of string facility made it difficult to do much more than a few ad hoc procedures in this area. Donald Knuth's I/O scheme, which came later, ingeniously circumvented the formal problems of I/O by list procedures. It was a very difficult scheme to implement; I spent a summer trying and failing. When General Electric did implement it for ASEA's new GE-625 it made the printer slower than the one running on their old Facit EDB. Moreover, of course, behind the scenes it required just the character variables and string handling we were so reluctant to add explicitly to the language.

Looking back, I regret the time we spent on polishing, standardizing, and subsetting Algol. Algol had played out its role, once they had implemented it in a number of sites and sown the seeds that were to give such a rich harvest of new languages. Making extensions where we deemed them necessary would have given us more benefit, in the short run and the long run.

2.5 Is there a *sens moral*?

With the benefit of hindsight, I can see that I took the wrong view in most of these discussions. I can only hope I did not seem too fanatic. Fanaticism clouds the vision and creates opposition; what one rams through in this way is seldom good enough and rejection will be its eventual outcome.

A more positive formulation of the same *sens moral* would be "Think, travel, and talk". One of the bad things about today's world is the impediments to travel such as expensive fuel, humiliating security controls, companies saving money the wrong way. Most trips pay for themselves in terms of knowledge gained and new ideas. Now and then, a single trip gives so much benefit that it pays for a decade of travel budgets. Cutting the travel budget in a crisis is a bit like throwing away map and compass when you are lost in the woods: it may lighten your burden but does not help you get out.

2.6 Why did our hardware efforts fail?

Why did our hardware efforts fail? Too little is known about what went on behind the scenes in government and in the companies involved. The easy way out is to say that it was inevitable. Much larger actors (Charles de Gaulle comes to mind here) failed, at much greater cost. Nevertheless, the fact remains that the Swedish government through MMN stumbled to the forefront of an extremely interesting field and let slip this chance, only to pump much more money into SAAB when it was too late. There cannot have been an ideological block or fear of technique in general: hundreds and thousands of millions went into nuclear power and even steel, and Sweden has a long history of making its living by innovation.

3. CULTURAL DIFFERENCES AS A CAUSE OF FAILURE

Perhaps we who worked in the field spent too little time talking to politicians, managers and the like. A few of us did, but most of us were so engrossed in making things work that we had little energy left for this long-range work. There was certainly no lack of conviction among us that this new gadget, the computer, would revolutionize the world. However, talking across cultural barriers is difficult. It takes a lot of experience to realize that we have to do it anyway, as the following examples try to show.

3.1 Technicians vs. administrators

In the early days, even elementary knowledge of computing was lacking outside our little band of specialists. For instance, when our managing director was going to inaugurate our compiler, he thought it was a piece of hardware and wondered whether it was transistorized. Not unreasonable – nowadays such special hardware is actually built – but at that time a serious lack of knowledge for one who had to make strategic decisions for our company. My point here is that we, who knew, did nothing about it. This was an awful mistake and one of the causes of failure. We might not have gotten through to our manager, but there is no excuse for us not trying. This kind of mistake – young experts looking down on their elders instead of sharing knowledge with them – is repeated daily with equally sad effects.

It is interesting to note that the company did not apply its strict routines for industrial secrecy to software development; we could cooperate freely with other institutions. On the other hand, they kept us out of hardware development; to this day I do not know what plans might have existed beyond building BESK copies and I/O equipment for them.

We had another very concrete problem in that the company wanted to use computers for office work, whereas we, their computer people, had received training in scientific and technical work. Few of us knew what an administrative office was like let alone how it worked. We never bridged this gap; we remained strangers in our company and were eventually pushed out to do what we knew best, as a service bureau. Our mother company never really found its place in this strange new world and foundered tragically some ten years later, after having been in business for some 500 years.

3.2 Professors vs. Computing centers

Yet another example of cultural differences, in this case between people of the same background and education. In 1964, the Swedish government's agency for administrative development, Statskontoret, decided it was time to organize computer resources for the universities. A scheme was set up which included fresh money earmarked for computer time, which the respective universities shared out to departments, which in turn passed it on to teachers and researchers. They could use the money to pay for computer services from any of the computing centers set up at the same time. The scheme also allowed the centers to earn money from external sources and use this money for extra computing equipment. To the rational people at Statskontoret this seemed a generous and flexible scheme of things. So, it

seemed to me, coming from the service bureau world to head the Lund center in 1968. It turned out otherwise.

The money allotted to the universities was for a one-shift operation of the centers. The idea was that the departments should find more money from other sources if the earmarked money was not enough – and very soon, it was in fact not enough. This went completely against the grain of university life. University people would not use hard-earned money intended for salaries to pay for computer time. Finding money for the computing center was the task of the head of the center, just as finding money for books was the job of the library director. The centers remained at one shift. A storm of protest ensued that finally forced Statskontoret to accept radically lowered night rates, after which our university customers rapidly filled three shifts. Meanwhile we had managed to find quite a lot of external customers, at some centers almost 50%. Now criticism took the opposite direction; they accused us of neglecting the university customers for the external ones. Eventually the centers decentralized as the advent of PCs made the old organizations obsolete. (After a dozen years as a teacher at the Computer Science Department, I now understand the academicians' side of the debate much better!)

The list of examples of cultural clashes could be long. Think of the lawyers who tried for decades to use copyright to protect program owners against unauthorized *use* of programs. Consider the standards institutes, organized on national lines, trying to cope with computing where the only national issues are character sets. Surely cultural differences of all sorts are so very important to success in a new branch that working place culture ought to be a subject in its own right at technical universities.

4. CONCLUSIONS

I would to like to conclude by giving my personal answers to two questions maybe all of us should ask ourselves at the end of the day: What is my greatest error of judgment in working life? I have to choose between two – equally embarrassing – alternatives: underestimating the impact of networks and the impact of the PC wave. Not that I did not expect the future to contain networks, but I expected a network to offer a much more standardized set of services and terminals, much like the telephone service used to be. I did not dream of people tinkering with their own computers on the present scale. A terminal, yes, and a printer, yes, and a friendly service man from the national computer service coming home on call to exchange non-working or too slow equipment. I dislike being told that my computer is ready for the scrap heap and that I have to go through all the cost and

trouble of switching to a new one, though the old one is sufficient for my needs. In other words, I expected (and wished for) a sort of computer service bureau on a national or global scale.

Does work remain undone? Yes, we still lack a standard, convenient, safe programming language in general use and a teaching and help service to go with it. Too much interest has come to focus on ready-made programs, which may be fine for the general user, but is far too restrictive in a research environment. It is as if we had invented printing and writing, but failed to educate people beyond the ability to read comic strips. The full potential of the computer will not be realized until the direct use of it is in every man's hands, a parallel to the alphabetization of the common man that took place in the 1800s. That is the challenge I would like to pass on to the next generation.

REFERENCES

In this paper:

[1] Peter Naur: *The European side of the last phase of the development of Algol 60;* pp 110-113 in Richard L. Wexelblat (ed.): *History of Programming Languages*, Academic Press, 1981, ISBN 0-12-745040-8

[2] Ingemar Dahlstrand: *A half year's experience of the Facit-Algol 1 compiler*; in BIT (Nordisk tidskrift for informationsbehandling), Regnecentralen, Copenhagen, Vol. 2 (1962), pp 137-142.

[3] M. Woodger: *Comment on a function designator changing the value of a formal variable*; Algol Bulletin 11.4, P. Naur (ed.), Regnecentralen, Copenhagen 1960.

[4] R. Bird & Ph. Wadler, *Introduction to Functional Programming*, Prentice Hall, 1988, ISBN 0-13-484197-2

Other books concerning Nordic computer history:

[5] Poul Sveistrup, Peter Naur, H. B. Hansen, Chr. Gram: *Niels Ivar Bech - en epoke i edb-udviklingen i Danmark* (Niels Ivar Bech - an epoch in the EDP development in Denmark), Copenhagen 1976, ISBN 87-980512-0-2.

[6] Jan Annerstedt et al.: *Datorer och politik - Studier i en ny tekniks politiska effekter på det svenska samhället* (Computers and politics - Studies on the political effects of a new technique on Swedish society); Zenith/Bo Cavefors förlag, 1970.