**To:** R.Brun, with copy to DPG

**From:** J.J.Bunn
**Date:** March 15, 1994

**Subject:** Comments on "Software in HEP: A strategy towards LHC"

Your paper entitled ‘‘Software in HEP: A strategy towards LHC’’ I found provocative reading, and I would like to make several comments on it. Perhaps like you, I have programmed in several different languages to varying levels of proficiency: Fortran, C, Prolog and Common LISP and others not worth mentioning. I am going to be challenging some of your statements about Fortran, so I wanted it to be clear that Fortran isn’t the only language I know!

My comments are distributed amongst the following sections of your paper:

- **Introduction**

  In the introduction you seem to be suggesting that it is somehow a bad thing that ‘‘... our software techniques and languages are not necessarily those used in mainstream industrial or telecommunication applications ...’’ whereas the problems we are trying to solve with our software are completely different in scale and scope from those addressed by software in these industries.

  The statement that the use of SASD was ‘‘... moderately successful ...’’ in a few experiments is understated in the case of Aleph, where components of the selected CASE methodology were and still are heavily used to make understandable the online and offline software. I particularly remember SASD review meetings in Aleph where various high-ranking members of the collaboration, who were not programmers, were able to make important changes to the algorithms being designed for the Reconstruction Program, simply because the SASD methods allowed them to clearly visualize the aforementioned algorithms.

  A ‘‘common simulation, reconstruction and analysis framework’’ may seem, on the face of it, a laudable goal for the LHC experiments. Of course, if there were only one LHC experiment then the goal would be easily attainable; one is left pondering the reasons why there are probably to be two LHC experiments. A clarification of what the framework referred to might look like to the end user would be useful.

- **Main components of the proposed software model**

  The bulleted second point states ‘‘The relative strength of Fortran 90 versus C++ will depend on the effort invested in the C++ training program.’’ C++ is as yet completely unproven as a viable language for HEP software: the onus is on the proponents of C++ to show that the language justifies the implied massive training effort you allude to. This has not been done. In particular, it has not even been shown that C++ is a better language than C itself for HEP. We are lacking testimonials from HEP groups who have tried languages other than Fortran for their offline software. In fact, we lack any firm evidence even from

outside HEP that C++ is a viable alternative to Fortran for large scale scientific applications[1].

The justification for putting effort into development of yet another memory manager like ZEBRA has to be made carefully in view of the availability of features in Fortran 90, C and C++ which seem to obviate the need for specialised memory manager packages. The need for an I/O package for efficient handling of storage of complex structures is more obvious. In any case, it is unclear what advantage OO methods have in this area.

The bulleted eighth point states that ‘‘The sequential file-oriented system used today to write raw data tapes should be converted to a data base hierarchical storage management system based on the direct access capabilities of ZOO’’, implying presumably that events are written by the online system onto non-sequential media like disks. It is hard to understand where the data capacity will come from. If, instead, the data are to be written to direct access tapes, then it is hard to understand how the necessary data rates will be achieved. In any case, an explanation of why sequential writing at high data rates onto traditional tape devices is inadequate is needed.

The ninth bullet talks of objects, but it is unclear if these are database "objects" or C++ "objects" or whether they are "entities" (i.e. objects) in the sense of the ER model.

- **Monitoring, Statistics of Usage**

  In Table 1, which shows PAW 1993 statistics, there are several entries for Falco terminals on Unix workstations. On the Apollo, for example, there are apparently 95 instances of use from Falcos. Exactly how do users invoke PAW from a Falco connected to an Apollo?

  It must be said that the monitoring data collected clearly apply only to the packages being monitored: they give *no indication* of whether those packages constitute ‘‘... the basic and common working environment ...’’!. And a clarification of exactly how manpower has been allocated according to results from these monitoring data would be interesting to see.

- **The Object-Based packages (RZ,HBOOK,GEANT)**

  It is offensive to imply that OO technology was invented by the authors of HBOOK when they decided to refer to Histogrammes by an integer ID. One can probably find ancient pieces of software that refer to individual tracks in a wire chamber by variables called "ITRACK", but to call these programs in any way OO is quite laughable.

- **The Interactive systems**

  There is a contention in this section that ‘‘The user interface is greatly simplified if the underlying package is able to pass messages to procedures....’’. It would be nice to see a justification for this statement: does it mean that coding the software for the user interface is simplified? If so, then we as software

[1] M.Metcalf received only four replies to a request on the C++ Newsgroup for details of large C++ applications.

developers should be careful about major paradigm shifts just because it makes our task easier.

- **Languages**

  The treatment of programming languages in the paper is inexcusably biased and subjective. It is hard to remain level-headed when trying to compose arguments against what is written, but here goes:

  - Mixing Fortran code with code in other languages, at the linker level, is no more difficult than mixing C with BASIC, Pascal with C++, or whatever. With the Fortran compilers I have used, linking with routines compiled with C (which is the most common requirement), has always been straightforward and well documented in the manuals.
  - There should be examples offered of codes (preferably useful in HEP) that are coded in C or C++ and that cannot be coded perfectly well in Fortran 90; which applications **need** to use pointers to functions? For these applications, is there no alternative? In what way are pointers to functions ‘‘essential’’?
  - The syntax of Fortran 90 is as easy to understand as the syntax of Fortran 77. The syntax of C++ in a real application is very hard to understand for programmers who are not used to the OO paradigm.
  - Fortran 90 is the best solution for large scale scientific computing today.
  - What is ‘‘... an automatic I/O package for all but trivial data structures ...’’ and why is it impractical to code one in Fortran 90?
  - It is stated that ‘‘Outside the HEP community there is decreasing interest and support for Fortran 90’’. If this is so, then why are the major vendors releasing Fortran 90 compilers? Why has Microsoft committed to an imminent release of a Fortran 90 compiler together with a fully-fledged SDK? Why did so many companies and individuals from outside HEP participate in the Fortran 90 standardisation process? If we talk about the language being used "out-there" for "most" applications, then the answer is not C++, not C, not Fortran, but in fact Visual Basic (or is it COBOL?).
  - To suggest that it ‘‘... may be immoral to force a language upon them that is totally useless in their future career.’’ when talking about HEP students learning Fortran 90, is misplaced altruism. Do we bother that an in-depth knowledge of the cross-section for phi production in proton-proton collisions will not stand these students in good stead when they walk out of CERN and into DuPont-Nemours as a VMS system manager? I think not.
  - On what basis is the claim ‘‘For software companies the development of Fortran 90 compilers is clearly a matter of low priority’’ made?
  - The section entitled **C and C++** continues the Fortran-bashing, and loses objectivity almost before it starts. It would have been more reasonable to make C and C++ into different sections, and avoid mentioning Fortran in either of them!

> The online community has always been in the pleasant position of being able to pick exactly the right language for the job, within the constraints of the hardware platform it was to run on. Hence the heavy use of online VAXes at LEP. The constraints for cross platform portability, adherence to standards and so on for offline software make taking what the online programmers choose as the offline language a rather hazardous affair. There is apparently no such thing as "standard" C++. How portable is my C++ program tested using the Borland C++ compiler, to the VAX running the DEC C++ compiler? Or should we all be constrained to the Gnu C++ compiler?
>
> The statement that ‘‘C is a very efficient language and not too difficult to learn’’ can be applied to assembler, as well. In fact, C is widely regarded as being a ‘‘high-level assembler’’.
>
> The example used elsewhere in the paper to illustrate "virtual functions" will make little sense to a non-C++ programmer: ‘‘... to introduce a new shape only the shape-dependent methods, which are declared as virtual functions of the base class, have to be written ...’’. It is said that the concept of information hiding will ‘‘optimize’’ the internal representation of an object, but optimize in what sense? Will it optimize the data size, or the speed with which the object can be manipulated? [1].

In conclusion, I believe the paper presents a biassed and unrepresentative argument in favour of a courageous departure from well tried and tested programming practises today in HEP. I believe the emphasis should be rather on

1. Picking the best programming language for a particular application,
2. Putting effort into ensuring that the Program Library is highly modular and easily and efficiently callable from client applications, no matter what language they are written in,
3. Easing and automating the program documentation task for programmers.

Despite all the above, thanks very much for such a thought-inspiring and provocative paper!

[1] The object could be optimised away completely, by recoding in Fortran 90!