The programming laboratory project.

The purpose of the project is twofold: to find the constituents of a better programming methodology and to apply them and to try them out in the design and controlled growth of a large, sophisticated system.

As target system I am envisaging a "programming laboratory", as I seem to need this for the education of software engineers. I regard programming primarily as an intellectual challenge and the full size and scope of this challenge only become manifest when we try to program something "large". But I cannot train future software engineers by only talking about large systems, for without first hand experience they cannot visualize, nor imagine what I am talking about. (This is precisely the programming problem: what happens inside the machine escapes our unaided imagination by several orders of magnitude!) The only way in which I can transmit what it feels like to partake in a many-man-year project is by letting them partake in it, be it only for a number of months perhaps. The advantage of the above target is its high demands on documentability, adaptability and extendability; it is my firm intention to give quite specific contents to a concept of "modularity" - a term which is often used as a motherhood statement- and expect the stated goals to force me to do so. Preliminary investigations, carried out during the last year or so, have convinced me that the design of an environment satisfying such requirements is not trivial; on the contrary, I have come to regard it as a worthy research object.

The kind of hardware I am looking for is a simple, very fast machine, with a reasonably sized primary store, a very large backing store, a line printer, a few keyboard terminals and paper tape peripherals and perhaps a single magnetic tape unit for safety reasons. Up till now the following considerations have been taken into account.

I am only too aware of the fact that I would never have been able to conceive the run time system for the X1 ALGOL 60 implementation if the X1 would have had built-in floating point arithmetic: if that had been the case I would have felt obliged to use it and the relative price of the system overhead would then have been excessive. That is the reason why I am looking for a piece of simple, straightforward but fast hardware: I intend to write in its machine code an interpreter for another machine, to do so once and then to forget the given hardware machine code. If the hardware machine is sufficiently fast, the resulting performance will remain acceptable for a laboratory environment, while it will mitigate the speed requirements for the backing store which must be very large. One reason is that at present the programmer regards (pre)documentation as an additional burden, among other causes because he cannot profit mechanically from it; one of the purposes of reshaping the programming activity is to find methods by which much of what is now called "documentation about the program" functions as an integral part of the program. Secondly I would like to do away with the off-line preparation of paper tapes or cards as part of the activity of program composition. This points to having the complete system documentation inside the machine, as readable as source text.

The keyboard terminals are viewed as "the programmer's desk". I need at least two of them, to create the environment of parallel interactions (extensions or modifications) with the same program, perhaps an "operators console" in addition to that. If the laboratory grows towards an environment in which larger numbers of students can get experience, more might be needed for reasons of more intense usage of the system.

My guess is that, particularly in the beginning, the main function of the line printer will be to act as a reproduction machine for the production of hard copies of (part of) the system information contained in the machines storage. Readability of the printing will then be more important than extreme speed.

The paper tape equipment is suggested as a means for getting started. I think that I would like to use the X8 ALGOL system to get off the ground. As the system grows, paper tape as back up system will be insufficient and a magnetic tape unit might come in handy.

* * *

Since the above has been written, I have had two conversations in Erlangen, relevant to the project.

One was with a number of people from Munich, one of which has been present at the NATO Conference in Rome in the last week of last October, where I had presented the quintessence and aim of the notion of "pearls". They were at that moment in the process of designing and implementing a tool for the construction of operating systems and could include at least a modest means catering for the "representational abstraction" as I had suggested. In Erlangen they showed me this with pride, joy and gratitude: they had already profited greatly from it. This was the second confirmation that the pearl concept seems to be a sound one (the first one came in a letter from Elliott Organick, but that was less elaborate).

More directly concerned with the configuration was a discussion I had arranged with Horst Huenke from Bonn, concerning keyboard terminals. I have been hesitating for a long time whether I needed printing keyboard terminals (teleprinter or input/output writers) or keyboards with a character display tube. For lack of experience with the latter I have been thinking in terms of the first for the last few months. In answer to my inquiry, Horst gave a strong plea for the terminals with character display tube. He warned me (himself) that his strong bias was undoubtedly also caused by

1) his extreme sensitivity to noises

2) the extremely poor input/output writer as coupled to an IBM 360.

But out of the discussion came two arguments in favour of the display unit, both consequences of the fact that the human eye is so well adapted to ignoring what it is not interested in. The arguments were the following (the first one I had thought of, the second one was new to me):

1) When one wishes to read something "in the system" -e.g. to look something up- typewriter speed is so painfully slow that one cannot ask for the "page" but must ask for the "line". This means that one must <u>know</u> the line and -if one indeed knows it- must also identify it via the keyboard. This is cumbersome.

2) Messages from the system must be more verbose for the novice than is needed for the expert. It is no solution to give your "degree of expertise" as this is not constant over the system: you may be currently an expert in one part but will no longer be one in a part which you have not been concerned with for a number of months.

In view of the fact that in the course of the project I hope to have students joining it, this aspect of human communication should not be ignored. I am therefore aiming at character display terminals. As Horst pointed out "A line printer is then an absolute must!"

So much for the configuration.

* * *

A central theme of the project is the similarity between system usage and system construction. The THE system was designed to bridge a preset gap -viz. from the given hardware to the five ALGOL-machines. Construction and usage of the system were at that time viewed as completely different activities, different in nature, taking place at different times and to be done by different people. This preset gap gave birth to a system of a constant number of "layers". In the mean time I regard the intermediate product, consisting of the hardware, covered by a few of the lower layers, as "a system" with its own right of existence and I regard the activity of adding a next layer no longer so radically different from what a user does when he feeds in his ALGOL-program into the completed system: by loading his program, he just creates a next environment, i.e. the environment in which his data are interpreted. The natural extension of the fixed set of layers of the THE system is a stack of layers, provided that we can add a layer which, in its turn, again can accept (manipulate and activate) a program, i.e. the next layer.

Acloser scrutiny reveals that "a stack of layers" is insufficient, we need a tree, because different environments have to be created in parallel. (In actual fact, this tree-structure can already be distinguished in the THE system, where at the top, five different ALGOL-programs may be running.) As "two" is the minimal degree of non-empty parallellism, at least two terminals will be needed. I trust that the tree-structure will not only be a conceptual aid, but that it will also play an essential role in the virtual storage management: the fact that each activity, by definition, only requires the nested environments to be found on its path to the root of the tree and that activities in parallel branches are by definition conceptually independent, is too fundamental a property not to be exploited in a systematic manner.

* * *

In our programming methodology we wish to do justice to the following points of view and observations.

1) A program should be regarded as the design of a large class of possible computations: these computations - the "making" of which is left to the machine - are the final product about which the programmer must make his assertions.

2) Correctness proofs are essential. Programming first and then testing is putting the cart before the horse, program testing can be used to show the presence of bugs, but never to show their absence. Program composition and demonstration of correctness are two activities that should be merged. (To quote A.G.Fraser: "I just want to make the point that reliability really is a design issue, in the sense that unless you are conscious of the need for reliability throughout the design, you might as well give up.")

3) The requirement that the correctness proofs can be given without undue amounts of intellectual labour calls for conscious exploitation of our powers of abstraction and a programming tool in the usage of which our abstractions can be suitably reflected.

4) A program should not be regarded as an object all by itself but as a member of a family of related programs, i.e. either alternative programs for the same task or similar programs for similar tasks; the function of the levels of abstraction is to indicate to what extent the different members of the family can be mapped on each other. We have to recognize the similarity between "the changed decision" and "the postponed decision".

5) The potentially large number of members of the program family considered requires that the correctness proofs are concerned with the family rather than with an individual member. This requirement gives a rather clear picture of the logical function of modularity; the programming tool has to allow (or even to invite) the corresponding "textual encapsulation".

* * *

In programming practice today, much attention is paid to documentation and one hears an urgent plea for pre-documentation. One is in trouble, however, as long as pre-documentation (a flow-chart, for instance) is regarded as describing intentions, as documentation about what is going to do, as it calls for the painful verification that the pre-documentation is still applicable. Rather than to regard pre-documentation as "statements about the program to be made" I would like to regard and treat it as integral part of the program itself, I want it therefore inside the machine, mechanically exerting its influence as much as possible. (And I want this "even at run time", provided that this remains a relevant concept: I want to come to grips with the problems that are presently described by "changing a program while it is running".)

The desire to have the different levels of abstraction not only reflected in the process of program composition but also recognizably reflected in the activities taking place during the course of the actual computations, will greatly affect the design. It is expected that this requirement will give a fresh appraisal of the operations which are presently covered by names such as assembly, linking, loading, binding etc. The desire to retain at run time more structural information can be expected to increase the demands made on primary storage (by a factor of two, say); this should be kept in mind while detailing the configuration.

One of my first concerns will be to find "a binding policy" (or rather: the basic primitives that allow us to exercise a class of binding policies). For this purpose I need a clear view of the degrees of combinatorial freedom called for by my concepts of modularity. Secondly I need a clear view of the requirements made by a virtual storage implementation. In order to complicate matters further -I remind the reader that it was my honest intention to do something difficult- it has been suggested that I should consider two levels of backing store (say a fixed-head-disc as second level store and a movable-head-disc as third level store). If this suggestion is followed, I shall not begin by paying extensive attention to the choice of an "optimum" strategy for the use of second and third level store, I would begin by implementing a straightforward one, the major part of my concerns being absorbed by the requirement that the replacement of one strategy by another would not be a major operation. As this strategy has to be implemented rather low in the system, this could easily be a very difficult problem, i.e. the kind of problem I should like to tackle. As a by-product we would eventually have a system in which we can easily experiment with different multilevel backing store strategies.

* * *

I should like to add some comments on my attitude towards "efficiency". To start with: I do optimize, but not purely on the usage of the mechanical resources, I do optimize, however, on "brainpower" because I regard that already now as our scarcest resource; with more and more powerful machines becoming more and more generally available, I expect that, in the years to come, brainpower will become still more markedly the bottle neck. It is for this reason that I am quite keen on program manageability, i.e. the ease with which we can effectuate the transition from one version to another -and such a transition might be desired for the sake of "mechanical efficiency". For certain design decisions -the hard ones!- it is vain to hope that one can postpone them or, when taken, can isolate their consequences; there will be rather basic decisions that will not allow to be changed at a later stage without overthrowing everything built on top of them. This is one of the undeniable facts of life. Here the only principle available is "minimum likelyhood of regret". From the past I know that in such cases I shall prefer the solution, the performance of which is the least sensitive to the future way of usage: I would rather decide to pay a constant price, "4" say, than the variable price "N" if keeping N low would impose an awkward burden on the programmer's shoulder or would impose too tough a scheduling task on the system (and its user population, for that matter!).

* * *

If industrial support for this project is offered, it should be seriously considered, particularly because the hardware needed is likely to have financial consequences which are somewhat unusual for the Department of Mathematics. Clerical complications that could follow from such a support should (in some way or another) be kept to a minimum; furthermore it should be made perfectlt clear to the supporting industry that this research has to be carried out without any form of secrecy. Without the explicit permission to talk and to write about the project to anyone I see fit, I shall certainly be unable to carry it out.

* * *