On a methodology of design.

On an occasion like this it is very tempting to look backwards, to play the modern historian, to give a survey of what has happened over the last 25 years and to interpret this history in terms of trends and the like. This is so tempting that I shall try to resist the temptation. Instead, I would like to pose a question and to speculate about its answer. The question is roughly "Can we expect a methodology of design to emerge in, say, the next ten years?".

Let me first explain why I pose this question in my capacity as a programmer. There is a very primitive conception of the programmer's task, in which the programs produced by him are regarded as his final product. It is that conception which has led to the erroneous idea that a programmer's productivity can be measured meaningfully in terms of number of lines of code produced per month, a yardstick which, when accepted, is guaranteed to promote the production of insipid code. A sounder attitude regards as the programmer's final product the class of computations that may be evoked by his program: the quality of his program will depend among many other things on the effectiveness with which these computations will establish their desired net effect. In this point of view a programmer designs a class of computations, a class of happenings in time and the program itself then emerges as a kind of generic description of these happenings, as the mechanism that evokes them. I regard programming as one of the more creative branches of applied mathematics and the view of a program as an abstract mechanism makes it perfectly clear that designing will play an essential role in the activity of programming. Conversely, the view of a program as an abstract mechanism suggests us that a good understanding of the programming activity will teach us something relevant about the design process in general. In actual fact this hope is one of the major reasons for my being interested in the task of programming for digital automata, digital automata who confront us with a unique combination of basic simplicity and ultimate sophistication. On the one hand programming is very, very simple; on the other hand processing units are now so fast and stores are now so huge that what can be built on top of this simple basis has outgrown its original level of triviality by several orders of magnitude. And it is this fascinating contrast that seems to make programming the proving ground par excellence for anyone interested in the design process of non-trivial mechanisms.

It is this combination of basic simplicity and ultimate sophistication which presents the programming task as a unique and rather formidable intellectual challenge. I consider it as a great gain that by now scope and size of that challenge have been recognized. Looking backwards we can only regret that in the past the difficulty of the programming task has been so grossly underestimated and that apparently we first needed the so-called "software failure" to drive home the message that in any non-trivial programming task it tends to be very difficult to keep one's construction intellectually manageable.

As soon as programming emerges as a battle against unmastered complexity, it is quite natural that one turns to that mental discipline whose main purpose has been since centuries to apply effective structuring to otherwise unmastered complexity. That mental discipline is more or less familiar to all of us, it is called Mathematics. If we take the existence of the impressive body of Mathematics as the experimental evidence for the opinion that for the human mind the mathematical method is indeed the most effective way to come to grips with complexity, we have no choice any longer: we should reshape our field of programming in such a way that the mathematician's methods become equally applicable to our programming problems, for there are no other means. It is my personal hope and expectation that in the years to come programming will become more and more an activity of mathematical nature. I am tempted to add that a development in that direction can already be observed, but I refrain from doing ~~so~~: such a statement has too much the flavour of wishful thinking and besides that, such a statement could easily be an overestimation of the relative importance of the rather limited part of the field that lies within my mental horizon.

I have used vague terms like "the mathematician's method" and "an activity of mathematical nature"; I did so on purpose and let me try to explain why.

In one meaning of the word we identify Mathematics with the body of mathematical knowledge, with the subject matter dealt with in mathematical theses, articles appearing in mathematical journals etc. I am not ashamed of admitting that most of it never passes my eyes; I also have a feeling that most of it -although of course one never knows- is hardly of any relevance for the programming task. If we identify Mathematics with the subject matter with which mathematicians have occupied themselves over the last centuries, it is indeed

hard to see how mathematics could be highly relevant to the art of programming. In view of the programming problems facing us now, we can only regretfully observe that preceding generations of mathematicians have neglected a now important field. There is of course no point in blaming our fathers and grandfathers for this neglect; In their time, prior to the advent of the actual computing equipment, there was very little incentive; for lack of machines programming was no problem.

In a second meaning of the word we identify Mathematics with a human activity, with patterns of reasoning, with methods of exploiting our powers of abstraction, with traditions of mixing rigour with vagueness, with ways of finding solutions. It is in this second meaning that I judge mathematics as highly relevant for the programming task.

It is perhaps worth noting that, at least at present, the second interpretation of mathematics does not seem to be the predominant one at the Universities. In the academic curricula the fruits of research are transmitted very explicitly, how one does do research, however, is taught only very implicitly, at most as a kind of by-product. We teach solutions, we teach hardly how to solve. At first sight this is amazing, taking into account that one of the assumptions underlying the University is that we can educate researches! But there are explanations.

One observation is that many mathematicians of the current generation -in Euler's time it may have been different- seem to worry very little about problems of methodology, stronger: they resist it and are shocked by the mere suggestion that, say, a methodology of mathematical invention could exist. Although we profess to be yearning for knowledge, insight and understanding, we are fascinated by the unknown and many a creative mathematician is fascinated by his own inventive ability thanks to the fact that he does not know how he invents. He enjoys his share in the spark of genius, untarnished by any understanding of the inventive process. We just like mysteries.

Secondly, with the growth of mathematical literature, particularly the publications of the type "Lemma, Proof, Lemma, Proof etc." mathematics has very much acquired the image of a "hard" science. It is regarded by many as the prototype of a "hard" science. But the result is that the mathe-

matician tends to feel himself superior, that he looks disdainfully down upon all the "soft" sciences surrounding him. As a result a serious research effort into discovery and development of a methodology of mathematical invention would have a hard fight for academic respectability. And we all know that the pressures for academic respectability are very strong. It takes a respectable scientist, supported by fame, to embark upon it. Polya did it with his "Mathematics and the Art of Plausible Reasoning" and I admire his courage.

The final reason why we teach so little about problem solving, however, is that we knew so little about it, that we did not know how to do it. But I honestly believe that in the last fifteen years the scenery has changed. Polya has written the book I mentioned, Koestler has written a book of 600 pages called "The Act of Creation", Simon delivers at the IFIP Congress 1971 a talk on "The Theory of Problem Solving", just to mention a few examples. And there is a fair chance that this development will influence our teaching of mathematics. I think it will.

After this digression we return to our original question, can we expect a methodology of design to emerge? In designing one designs a "thing" that does "something". Over the last decades the most complxated things designed to do something have been programs; on account of their abstract nature we can regard programs as the "purest" mechanisms we can think of and if we can find some sort of question to the specific question "What about a programming methodology?", that answer seems relevant with respect to our original question.

It is my impression that there is a point in discussing programming methodology separate from problem solving as it is treated usually. Most of the literature about problem solving that I have seen deals with how to hit an unexpected but simple solution -simple of course once you have found it. In the case of programming this simplicity of the final solution is very often an illusion: programs, even the best programs we can think of for a given task, are often essentially very large and complicated. And by structure they are more akin to complete mathematical theories than to an ingenious solution to some sort of combinatorial puzzle. In other words, programming tasks seem to be of a different size.

From a programming methodology we require two main things. It should assist us in making better programs -i.e. we have desires regarding the final product- it should also assist us in the process of composition of the design -i.e. even if we have established what kind of programs we should like to design, we would like to discover ways leading to such a design. As it is hard to talk about strategies that might assist you in reaching your goal without having a clear picture of the goal itself, we shall deal with the first question first: what kind of programs should we like to make? If we talk about "better programs", what standards do we apply in judging their quality?

I have raised this question urgently and repeatedly in the first half of the sixties but at that time it turned out to be impossible to reach even the illusion of a consensus of opinion and the question was discarded by the attitude that it was all a matter of taste. The common experience of the next five years has certainly changed the situation. This common experience with large and vital programs was very often disastrously bad, and as a result of this sobering experience more and more people agree that requirement number one is not only that a program should be correct, but that its correctness can be established beyond reasonable doubts. An analysis of the possible ways for increasing the confidence level of programs has shown that for that purpose, program testing must be regarded as insufficient: program testing can be used very effectively to show the presence of bugs, but never to show their absence. Proving program correctness remained as the only sufficiently powerful alternative. And here I don't necessarily mean "formal proofs": I regard axiomatics as the accountancy of mathematics, or to use another metaphor: a formal treatment relates to my power of understanding as a legal document to my sense of justice.

The concern about correctness proofs had an immediate consequence. If proofs get longer and longer they loose their convincing power very, very quickly. It also emerged that the length of a correctness proof could depend critically upon the structure of the program concerned and with this observation a legitimate objective of program structuring emerged, viz. to shorten the length of the proofs required to establish the confidence in the program's correctness. Such considerations gave rise to a computing science folklore for which I am partly responsible; it centers around key-words such as "hierarchical design" and "levels of abstraction".

The programs should be "correct", but that is certainly not the whole story. Correctness proofs belong to "hard" science - and the more formal the proofs, the harder the science. Considerations about the relation between program structure and proof length are already at the outskirts of hard science. Softer still is the equally vital requirement that the program, the mechanism in general, be adequate, that it be a sufficiently realistic implementation of the model we have in mind. Let me explain this with a simple example. In ALGOL 60 the integer variable is a key concept: whenever it is manipulated it stands for an integer value, but in understanding a program, you don't care about its specific value, you have abstracted from it. Caring about its actual value is something you leave to the arithmetic unit, you yourself understand the program in terms of variables and relations between their values, whatever these values may be. In order not to complicate matters we restrict ourselves to applications where integer values remain quite naturally within the range that might be imposed by the implementation. Suppose now that our machine has a very funny adders, funny in the sense that each integer addition takes one microsecond except when the sum formed happens to be a prime multiple of 7, in which case the addition takes a full millisecond. How do you program for a machine like that? You might prefer to ignore this awkward property of the adder, but if you do so I can change the machine, slowing down the exceptional additions by another factor of thousand, and if necessary I do so repeatedly. Comes the moment that you can no longer afford to ignore this awkward property: by that time you feel obliged to organize your computations in such a way that the exceptional additions are avoided as much as possible. Then you are in trouble, for a vital abstraction, viz. that of an integer variable that stands for an integer value but you don't care which, is denied to you. And when a vital abstraction is denied to a user, I call the implementation inadequate.

The requirement of adequacy has a direct bearing on out hierarchical designs, more precisely on the number of levels we can expect to be distinguishable in such a design. Mind you, I am all in favour of hierarchical systems, we have hierarchical systems all around us. We understand a country in terms of towns, towns in terms of streets, streets in terms of buildings, buildings in terms of floors and walls, floors and walls in terms of bricks, bricks in terms of molecules, molecules in terms of atoms, atoms in terms of electrons and nuclei, nuclei in terms of what-have-you etc. It is a pattern you find all over the

complete spectrum ranging from science to children's behaviour. At each next level, however, we describe phenomena of a next order of magnitude. In the example given it is a spatial order of magnitude, in the case of mechanisms were we want to understand what happens, we find ourselves faced with happenings to be understood in different grains of time. It seems characteristic for an adequate design that when we go from one level to the next, the appropriate grain of time will increase by an order of magnitude. If this impression is correct, our adequacy requirement imposes an upper bound on the number of levels admissible in our hierarchy, even if we start at the bottom at nano-second level. Then we must conclude that, although essential, hierarchical levelling cannot be the only pattern according to which "Divide and Rule" is to be applied.

Now the design process itself. Many of its aspects can better be treated by greater experts in the field than myself; let me confine myself to what I have found lacking in the literature. On the one hand you find authors writing about problem solving: they stress psychological conditioning in order to hit the unexpected solution and search strategies. Their descriptions of the inventive process seem honest, their guidelines seem relevant, but they confine themselves to small size problems. On the other hand I have met people trying to organize large scale design projects. They were mostly Americans and talked with the self-assurance that we tend to connect with competence. I am perfectly willing to admit that once or twice I have been taken in by their eloquence, but never for long and I have come to the conclusion that the organization expert, although potentially useful, will not provide the final answer. A few things have struck me very forcibly. Firstly, they persist in thinking in exactly two dimensions - this must have something to do with the two-dimensional paper on which they draw their organization charts. Secondly, they are obsessed by reducing the elapse time; this gives them the opportunity to introduce their dear tools such as PERT diagrams, base-lines etc. but I am much more interested in the designs we don't know how to achieve even if we are not in such a great hurry. Thirdly, they have such preset notions about documentation standards and the holy rules of the game -such as design reviews- that the whole design efforts loses the ring of reality and degenerates into a complicated parlour game. Buth the fourth thing is probably the worst: apparently they do not know the essential difference between "vague" and "abstract" where it is the function of abstraction to create a level of discourse where one can then be absolutely

precise!

Let me now give you what I regard as my expectations. I leave it to you to decide whether you prefer to regard them as my hopes.

Our insight in the effectiveness of patterns of reasoning when applied to the task of understanding why mechanisms work correctly and adequately, has been growing considerably in the recent past and I expect it to grow still further.

Our insight in the design process will also increase. In particular I expect that more recognition will be given to the circumstance that designing something large and sophisticated takes a long time. As a result we must take the intermediate stages of the design into consideration and must be clear about their status in relation to each other and to the complete design. I expect a clearer insight in the abstractions involved in postponing a commitment.

From a better understanding of the relation between the final design and its intermediate stages I expect a body of knowledge that will enable us to judge the adequacy of descriptive tools such as programming languages.

In the course of the design process we are envisaging a final product: how well it behaves will ultimately XXXXXXXXXX only be known by the time the design is completed and the mechanism is actually used. By its very nature the design process makes heavy demands on our predictive powers. In connection with that I expect two things to happen. On the one hand our predictive techniques will be refined: at present, for instance, the outcome of simulation studies tends to be the source of heated arguments and it appears that we can simulate but lack the proper discipline that tells us what weight to attach to these simulations. Refinement of predictive techniques is one thing, the other thing I expect is that we shall learn how to reduce the need for them. In the design process it is not unusual that some sort of performance measure is dependent in a complicated and only partially known way on a design parameter whose value has to be chosen. There are two usual approaches to this problem and they seem to be equally desastrous. One of them is to give a group the duty to discover the best value of the parameter. As they don't know how to do this, any answer they produce will fail to be convincing and as a rule this

approach leads to heated arguments and an overall paralysis of the design process. The other approach leaves the parameter free, so that the user can set it, suited to his own needs. Here the designer has shirked his responsibilities and leaves to the user that part of his task that he could not do himself: this second approach is disastrous because often the user is equally unable to fix the parameter in a sensible way. Both approaches being equally unattractive I expect the competent designer to become more alert when the problem of the parameter with unknown optimum value presents itself. The most efficient way to solve a problem is still to run away from it if you can, and one can try to restructure a design in such a way that the parameter in question looses its significance. In individual applications the performance might be less than optimal but this can be easily outweighed by greater adequacy over a wider range of applications and the easier justification of the remaining decisions. This is an example of the impact of the requirement of "designability" upon the final product.

This was a very rough sketch of a few of my expectations of an emerging methodology of design. I am not going to refine in this talk the picture any further for part of my expectation is that further refinement will require the next ten years. But by that time I expect a body of teachable knowledge that can be justly called "a methodology of design". Other authors are less modest in their expectation: Herbert A.Simon argues in his little booklet "The Sciences of the Artificial", which I can recommend warmly, that what he dares to call "a <u>science</u> of design" is already emerging. He may very well be right; personally I feel that I lack the wide experience needed to judge his prophecies.

I would like to end with a final remark in order not to raise false hopes. The remark is that a methodology is very fine but in isolation empty. We expect a true methodology of design to be relevant for a wide class of very different design activities. The counterpart of its generality is by necessity that it can only have a moderate influence on each specific design activity, i.e. we must expect each specific design activity to be heavily influenced by the pecularities of the problem to be solved. Ans that is where knowledge about and deep understanding of the specific problem enters the picture. Yet a methodology, although absolutely insufficient in itself, may be of great value. It should give as the delimitation of our human abilities, it could very well result in a modest list of "don't"'s, rules that we must obey and can only transgress at our own peril.