# IS THERE A "BEST" PROGRAMMING LANGUAGE FOR DESIGN AUTOMATION?

Moderator: M. Weindling

Panelists: R. Mandell
J. G. McKinney
P. H. Dorn
D. T. Ross
P. Semple

DOUG ROSS, MIT - Electronic Systems Lab

When I saw the title of this panel discussion, "Is There a Best Programming Language for Design Automation?", I thought that this would be the shortest remark I ever had to make - one word, AED, and then I got to thinking about it and noticed that wasn't really keeping to the subject of the panel, so I had to double the length of the paper and say, "Yes, AED."

I really don't feel terribly controversial yet this afternoon, so I would like to be very brief and merely try to put a slight twist, or maybe an elaboration on the title of the panel, because I think it is ambiguous as it's stated and we should spend some time trying to remove this ambiguity.

Mainly, are we talking about a programming language for making a design system or a programming language for a designer to use in doing design. It seems to me that the key point is that it must be that we are to be talking about the design system building language because I've been listening to various of the talks given here and also ones given over the past several years other places and I think that the contention that we have had from the beginning of our interest is computer-aid design, which goes back quite a ways, is borne out-mainly, that what is needed as far as the users of design systems are concerned is not a single best language, but a very large number of specialized languages, so that each kind of user can use his own "shop talk jargon," needknownothing about programming preferably, etc., etc. So, I think that we must be talking not about languages for the designer to use, but, rather, languages for building design systems.

Then, too, if we now are going to be talking about languages for building design systems, I think that there, too, is another layer of ambiguity--potential ambiguity--or a layer of things that need to be clarified. Mainly, are we talking about just a language, or are we talking about more than just the language and here again, my contention is that it must be that we talk about more than just the language, mainly a language embedded in a system or systematic way of using the language. In other words, what you write on paper itself is not the important thing, it is how you go about using the language to create a design system.

A couple of years ago--I guess it is almost two years ago now--somebody used a term and I wish I could remember it because I would like very much to acknowledge it. They used a term in connection with programming languages which struck me as being very important and, in fact, is the place where I thought this afternoon's panel discussion really should focus, and that was with respect to a programming language there is a culture that goes along with the use of that language and I think that this is the real key point--is the culture the whole method of using the language of creating systems, and so forth. This is the central point if we are to make much headway in making design systems without everybody going broke in the process and, also, incidentally, hopefully meeting some software timing schedules.

I'd like to just sort of stop at this point by saying that this view has been behind all of our activities in the computer-aid design project at MIT since we started in 1959. We said in those days, the way we put it was we couldn't possibly learn as much about design as all of the various people in industry, all the actual different types of designers and all the different design groups within various companies, so that the only thing that we could hope to do if we were to play any role at all in this area was in working on the underlying methods of making design systems, getting some deeper understanding of what it is to solve a problem and from the very beginning this philosophy has colored all of the steps that we have taken, which nowadays are in the form of concrete working systems of a family.

I would sort of like to leave my initial remarks at this stage, merely having tried to clarify what my view is of the panel topic and see what the response is from the other panel members before going into any greater detail at this time.

DICK MANDELL, UCLA

I think I would have to agree with Doug that there are definitely two types of languages here. I would like to address myself to the one in which you build a design automation system and, again, I have to agree that there are many levels and I would like to talk about the lowest level of the language which you would like to use, namely, the language which you use to program these other special programming languages which you use to build a design automation system.

I think in evaluating any language of this type, there are several things that must be considered, and they are very difficult to consider. It can only be considered in a very general way.

There is, first of all, the question of how easy is it to express the things that you want to express. How easy is it to learn the language; how easy is it to debug the program. Then after you have decided this, you have to now look around and see what's the quality of the implementations that are available. That question leads you to when you begin to use the language and, of course, when you are talking about implementation, you've got to ask about compilation speed, execution speed, the kind of error messages you get, the efficiency with which the implementation uses storage, and the stability of the implementation over a period of time. Is the language going to change so much that the programs you wrote a year and a half ago aren't going to work. And this stability question also raises the question of what happens when you change machines and probably rules out assembly language, except in the most dire cases.

Now with that in mind, we go back to the question of what do you have to express. I think that there are a number of things which are at least characteristic of design automation and at least some of those are the fact that you need very flexible I/O. You really do a lot of character and bit manipulation and since you have a lot of storage, you like to be able to pack these things as tightly as possible. Very frequently, the nature of these kind of programs leads you to list processing. You also want to build compilers for the special program languages in which you really express the work that you are doing, so that you need a facility for developing special purpose languages. Finally, you would like some very good control over the way storage is allocated.

Now, the language which I would like to propose meets at least a good part of these numerous things is PL 1, augmented with a language which we are using at UCLA which was developed at SDC called Meta-5. PL 1 has the ability to manipulate characters and bit strings; it' s fairly easy to de-bug because there are some de-bugging statements in the language; it allows a large number of ways of obtaining access to random access devices and getting the things off of them that you want. There are a number of ways of generating printed outputs, some very easy that can be used in de-bugging and others a little more complex that can be used for obtaining very nice looking output, and it provides the recursive procedures which are very useful in a number of contexts.

Finally, you do have control over storage. Now, the language does fall short at the present time in the area of list processing, but one can make up for this in the Meta 5 portion which I would propose and, also, I am quite confident that eventually there will be a good list processing facility in PL 1. Now I should point out that these things I have said are also present in Doug' s language and I wouldn"t like to take the position that I'm saying that I definitely want to choose PL 1 as opposed to AED, but I have found PL 1 to be extremely useful; the learning problem is not too severe because you can learn a sub-set of language and piece in the things that you need. Again, it' s very easy to express and the system de-bugging aids and messages are very good.

## PARLAN SEMPLE, IBM

Since my own specialty of generalized systems is probably the most removed from the very specific field of design automation, I would like to limit my remarks proportionately.

Subject to that qualification, however, it is probably not possible to have a best language for two very important reasons. The audience gathered here today, I am sure, if surveyed concerning their past experience and present work in the field would undoubtedly reveal a wide diversity in both background and application areas.

Concerning the most important person first--theuserof any language--his choice of an optimum vehicle for communicating with a computer will be dependent probably primarily upon his own education and previous experience, particularly in this industry where multi-million dollar computers are almost obsoleted by the expanding technology as rapidly as the family car. Perhaps that's a little bit of exaggeration, but we're almost there. We probably all tend to transfer knowledge between similar concepts. To the degree that we can make this transfer between similar concepts, the less new knowledge that we have to learn in time which we never have. As a result, each individual user tends to stay within the software family at least in concept. Now, regardless of the language which an individual user might nominate as being best, and probably this nomination will be dependent upon his previous experience, his favorite cannot possibly encompass the broad realm of applications.

I have seen this in even the brief review required for my own paper. When the design engineer reaches this impasse, he will have three choices. One, he will try to find the language that does have the missing features. Two, he will probably try to write a specialized subroutine himself and then branch to it. Or, three, if at all possible, he will employ his favorite programming language in a way it was never visualized to be used. My own paper is an example of that last category.

But with respect to generalized languages, such as GIS, being best, I suspect that there are some application user combinations (and you notice I use these two terms together) where this will occur. Looking into my handy computerized crystal ball, the probable applications area for systems like GIS--notice I say like GIS, since there are many involved--will be situations where the problems are fairly dynamic and their solutions equally transient. Hence, for those operational problems demanding flexibility, generalized systems, even with a probable inefficiency, should be the most economical solution in the long run.

With respect to the other and most important half of that application user combination, I suspect that GIS users will be those design engineers of today--the people who are most familiar with the decision tables or data basis which they are involved with right now.

Although some of this logic appeared to be a little involved for those of you who saw my paper, most of the people in this room could achieve a working understanding of the GIS language, as an example, probably within less than a week.

To summarize, then, nevertheless my vote will second those on the panel who believe that there cannot possibly be a best language because of the very diverse backgrounds and applications of the personnel and their work in the field of design automation.

PHIL DORN, UNION CARBIDE

My own feelings, regrettably, from the viewpoint of controversy at least for this discussion, very much agree with Doug Ross's and some of the feelings expressed by Dick. I have been inclined to believe that in any design automation in operation, we have approximately four levels of programming going on and I think I have a fairly good answer to what is the best, with quotes around it, language for at least the very top level. The top level, of course, is the user who is probably not a programmer, 99 percent of the time at least. The best language for that user is the language with which he is most familiar and most comfortable. If he is an engineer, it is probably the things he learned in engineering school; and he could be an accountant, or a bookkeeper, or almost anything else. The language must be geared to the user. It cannot be a programming language as we in this room know it. I'm fairly sure there is a best up there, but I haven't the foggiest idea what it is.

We want to drop down one more level and reach some of the programmers--the people who are actually doing the really hard work; at least that's what we think. We can select any one of a number of procedure-oriented languages. PL 1 would be one candidate. I have some doubts about FORTRAN--I don't think it's rich enough to do the types of jobs that we want to do.

We want to drop down a third level and reach the poor, downtrodden systems programmers. There are really two particular kinds of programming going on at the third level, as I see it, one of which is essentially old fashioned systems programming--bit picking, handling the I/O, handling the relationships between a design automation system and the larger operating systems of that machine. He may wish to use assembly language; he would probably be better off if he could use something along the lines of PL 1. Here again, I don't think FORTRAN gets close enough to the operating system to allow the skilled programmer to really do the job without slipping into assembly language. Also down at this third level--in a design-automation environment, at least--somebody has got to be worrying about two additional subjects: communication with consoles and data bank access. It is clearly a preposterous notion to have every other programmer in the place worrying about console communication codes in an interactive environment. One or two people at best should be able to write all the packages. What language to do this in I have no idea. Up to now we have been doing it pretty much in assembly languages.

Another very major subject which has to get handled somewhere is who is going to control the access to the data bank. It's probably down at this level. It should not have to be done in an assembly language. The one design automation system with which I am very familiar, was done in an old, tired procedure language--MAD. We're pretty sure that this type of thing--skipping through a data base--can be done in a compiler language. So we've really got many levels--many languages--all connected into one system. There is one thing I am fairly sure of in this whole maze of languages. Anybody who will seek to give the user and the applications programmer at the next level down--the second level programmer--a system in which he has to use assembly language is barking up the wrong tree. I'm ninety-nine and forty-four one hundredths percent sure that there is no room for assembly language at the top level. I barely will tolerate it at the lower levels. The other major problem which I really see in front of us is this data handling problem. I'm sure Doug Ross will have more to say about that. It's the key problem to all of these systems and in the course of this week--Monday and Tuesday so far, at least,--we have slipped by the data problem. Yet it is terribly, terribly crucial.

I have a few more noncontroversial remarks, but I think I'll save them up for later. I'd be interested in hearing the views of the rest of the panel, particularly on how does one handle a large data base cleanly, efficiently and essentially concealed from the applications programmer who doesn't really care how you get to it--just get it.

## ALAN STONE, HUGHES AIRCRAFT

I don't have any prepared remarks because I'm sort of a last minute fill-in here, but let me just throw out some disconnected thoughts. It is my impression that there are many design automation activities in many companies that have been sorely burned by the conversion problems of their original assembly language first and second generation systems. Many of these shops are now artificially restricting themselves to either FORTRAN or COBOL that would be the only thing available to them. Neither of these languages is really entirely suitable and this seems to be a transient period, with many companies watching and waiting and working only at partial efficiency because of these language restrictions. I feel that because of the implementation difficulties in PL 1 it will be some years yet before it becomes as standard as FORTRAN has become for widespread applications. In the meantime, I think we will see that vacuum filled by further language designs. I'm talking now about not problem oriented design languages, but languages for developing design automation systems. I see in other areas for special form applications, real time programming, there is work under way developing standard special purpose programming languages that look somewhat like PL 1; that have list processing and real time extensions to them. In many cases these are very rich, complex languages and will certainly run into the same implementation problems that PL 1 has. But because of the lack of complete acceptance and availability of PL 1, people are sitting down designing new languages. I believe there is an effort going on sponsored by the Air Force--an another one sponsored by NASA, completely independent--on real time special purpose programming language design.

Languages that already exist, such as the AED system, my personal feeling is that these will not be--their acceptance will be limited because many industrial concerns will be reluctant to commit themselves to a programming system that's sponsored and supported by universities.

DOUG ROSS

I must respond to Alan--that's the first time I ever head of the universities being placed in that light. Also we would like to think that all we are doing is serving as a nucleus, and we would like very much to get out of the sponsorship racket if we could only get our industrial friends to pull together long enough to shoulder more of the load.

DOUG ROSS

I think the original points that I raised have been well taken by the remainder of the panel in that it does seem that we are in agreement about the languages for the user being not one of a kind but one of many kinds to suit the user. But I would like to come back to this matter of the "best" language as being really not just the language but the whole system and the whole way of operating. I have a paper that will be in the ACM national meeting in Washington. The proceedings there, in which I go into some depth about the whole approach of our AED effort which is a great deal more than just the AED-0 language. This is one problem we have in communicating with people. The AEO-0 language was never properly designed. The reason it has a zero on it was because we intended to use it only within the project and not let it be used outside. So it was sort of thrown together based on things that we knew we needed, taking Algol 60 as a base, but it has never gone through a true, proper design stage. I think there is some question about whether the design stages that the other major language which has been mentioned here has gone through--whether those were properly designed, too. But that is not the point. The point is that the language--as a language AED-0 has not been effected even in minor ways since 1964. But this doesn't mean that we've stood still with respect to the subject matter of this panel. Instead, what we've been doing is concentrating on the very building blocks that must go into creating any sort of a design system; doing these things over and over again.

Many of our packages of routines for handling these various topics which have been mentioned have been reworked over a dozen times in order to get better efficiency, better modularity, more machine independence and so forth, and the way we like to look upon these packages of routines that are the building blocks out of which you build other systems--we like to think of them as the semantics of a high level "best" language which is not yet around, but we are working on the semantics of that language without having the syntax--without yet having it in the form where you can write nice programmer-user oriented phrases for these things. But we do put into the hands of the user the power and expressability and flexibility of these very advanced linguistic techniques. An I think that this is where the other systems are behind the need with respect to making computerated design systems, in that it's not sufficient just to have a compiler; it's not sufficient just to have a language for the programmer to write.

What we need to be doing as a group--not just us but the whole industry--is concentrating on where the essential tough parts of building computerated design systems, how can we deepen our understanding of those tough parts and reduce those to the form of working programs which have sufficient modularity so they can be put together in many ways to move the many different systems that we need. And this is where I think the effort we have in our work is in some ways--I'm stumbling around because I don't want to say that it's more unique or advanced or what--it's just we've been at it longer and I think it's an area that too many other groups are not focusing on well enough--they're too interested in the particular job that they're trying to do right then. And I think the problems of making design systems are too big to be solved just by Yankee ingenuity or by just making a better mouse-trap. We have to deepen our scientific understanding.

Let me take just one example of this to illustrate what I'm talking about, and that is this matter of use of storage in the system, not only on a mass data basis but also at the fine level right into the innards of problem modeling. Now it has been mentioned that PL-1 is weak in the list processing area. Well, I don't think this is going to be satisfied by adding to PL 1 a list processing capability. There is in the most recent issue of the ACM Communications a description of the currently available things in this direction.

I think there's a more basic problem in that the real key in storage management--whether on a fine scale down at the word and bit level all the way up to the mass storage level--is that no matter what is stated in the system, no matter what part of the system you're talking about, some part of its behavior is going to be based upon the use of storage.

I feel very strongly that the question of use of storage is paramount and should come before everything else. Now this means that you cannot take a language, design into the interpretation that the users will give of that language all sorts of very exotic storage controls, and expect that that is going to do all of the proper things for all possible users. Instead, you should have the use of storage implemented in terms that can be both built into the system automatically and be available to the user.

Now this means that the storage handling system cannot be buried in terms--I'm having difficulty expressing this because I can't remember the terms that are used elsewhere--but they can't be buried behind single words in a high level language in which the user is not really sure of what's going to happen. Instead you should have the free storage system first and then use that system as you implement both the compiler and the compiled programs and the systems which are represented by the programs that are compiled.

Now I think that until this is recognized we have a lot of confusion about what is the difference in implementability of these languages, efficiency on different machines, etc. I think we must take these major building blocks out of these languages, efficiency on different machines, etc. I think we must take these major building blocks out of the innards of high level languages, look at them as basic steps that are needed in using the computer, and then go back to our business of designing systems.

Here again there is our view on the free storage question. I just put in the mail this morning the proof copy of an article for the August Communications which will be the description of the free storage package that we have. We'd like to have this serve as a kick-off point for more in-depth discussion of this important area. That's just one of the things that I think is the trouble with making systems--it's got to be more than the language, it's got to be what are you really doing.

DICK MANDELL

I think I'd like to answer Al's comment about the implementation difficulties of PL 1. I think they stem from two things: (1) when the first release of PL 1 came out it had a lot of bugs in it, and we simply had to program around them. A lot of people heard about this and a lot of people got very upset about PL 1. I think that we have to admit that PL 1 is not the fastest thing in the world and it doesn't use storage the best way it could. But on the other hand, I can program about two or three times as fast because I don't make as many errors as with the older languages. I find that the diagnostics are very good and that the ability to say things which I want to say is there.

The other problem is that one has to gain experience in a language like PL 1. You're very far removed from the machine and very frequently what is easy for you to say is not the easiest thing for the machine to do, and you just have to perform some timing experiments and fool around with it until you find out the really right way to use it.

The other problem is that you see a lot of hesitancy on the part of other manufacturers than IBM to implement the language. They say it isn't fully settled down yet, but I think the fact that there is a living implementation of the language somewhat refutes that. That compiler almost acts as a standard. The only way that manufacturers are going to be convinced to make PL 1 available is for people to demand it. And the only way for people to demand it is to find it, try it and see that it is a good language.

PARLAN SEMPLE

Although Phil and I didn't exchange notes ahead of time his following me worked out very nicely. I really liked his classification of users, for this introduces my comment to the gentleman on my left here--for whom are these programs and systems being developed?

If we are considering the top person who is not a programmer, he could care less where his program is residing in storage. The fact that it's on this device or that--leave that to the operating system. There's a similar comment about efficiency. Remarks were made--and again I don't mean to a particular application here--only to the extent it illustrates a concept. Here we have references to reiterative improvements, increasing efficiency, but frequent changes.

So again, in terms of total cost--in terms of getting the job done for that number one user--what benefit has it been to squeeze out the last micro-second, because within two weeks or some other relatively short time the program is going to be changed anyway.

PHIL DORN

I'm very glad to hear Dick do the dirty work I usually have to do. I can't help but agree with Doug in a general summary of the problems of the field, although I personally think the data handling problem is infinitely more severe than the storage management problem. This may be a personal view. I think they are both very, very nasty problems, which were, at least in the data handling area, in spite of some of the work that has gone on in your part of the country, (Larry Robertson's work, and yours), I don't think we are anywhere near a solution yet. I'd love to say within the next three or four years, but I suspect it's really further away than that. Maybe you've got the free storage problem licked, or maybe we just need bigger machines. As my favorite friendly IBM salesman always says, "buy more core." It's a standard answer.

I again want to get back to the question of the so-called "best" language for the user of the system. I can't argue too much with Doug's approach as to building the system, but I'm much more concerned personally with the user. I've been in an environment where the user is really a guy who has never seen a computer, and we have had in fact cases of people working at a console graphics system that have gone on for three or four months without ever going around behind the wall and looking. Gee whiz, there's a computer back there. They couldn't care less. The language which they deal with sort of looks like a programming language, only it's a language in their terms.

In the particular case with which I'm very familiar was a language for automotive body designers who talked in terms of lines and points and sines and cosines (which incidentally they do use). They don't talk about wheels and trunks. The trick of the whole thing was giving them the language with which they were most comfortable. In fact, what we did was let them design it in a very peculiar sort of a way. We sat down and lived with the user for three or four months. How does he talk? How does he think? Does he understand what we in programming have known as an iterative procedure for 10 years? It turns out we allowed them to invent a loop. They thought they had invented the wheel. This was great. It was many years afterward before we told them we had been doing it in FORTRAN and the machine languages for 15 years. We allowed them to work their way through the problem.

So what do we need to produce this kind of language rapidly? That seems to me to be one of the key questions. Maybe the AED system will do it. I think Meta 5 will allow the writing of a compiler for a user language rather rapidly. But, even if you get the user language defined--and this can be as little as 2 months, I suspect, in a situation which is not really too complicated, at least from our viewpoint, such as the bookkeepers, the credit people with on-line systems, we could probably design a language very rapidly. Where is all the rest of that system underneath it coming from? That better be there when you design your top level language.

ALAN STONE

We haven't had much audience interaction yet, but my own interest and edification--most of the audience is either directly involved in implementing design automation systems or close enough to it to know the details of what's going on. I'd be interested in hearing by a show of hands what--how many people are using FORTRAN for this purpose. I'm talking about implementing systems, not as a user language. What shall we start with?

| | |
|---|---|
| FORTRAN | 85 percent |
| Machine Language | About 50 percent |
| Cobol | About 5 percent |
| PL 1 | About 10 percent |

DICK MANDELL

Meta 5 is a language that SDC developed some time ago as a research language for writing compilers. We picked it up and programmed the compiler in PL 1 and in Meta 5. It has the PL 1 part and the Meta 5 part which allows you to bootstrap and make continual changes. We should be submitting that to SHARE within a month or two. We currently don't have a manual out, but a manual can be obtained from SDC. Unfortunately I couldn't bring a number along with me today, but I'll have them tomorrow if anybody is interested.

(Discussion opened to the floor for comments and questions)

DON PARKER, CONTROL DATA

Without fear of acting as the straight man for Doug Ross, I wonder about the concern of computer installation managers and the fear that this panel may be throwing into their hearts with the agreement--general agreement on a mass proliferation of programming languages. These managers who are already worried about the four or five hundred or more programs that are in obsolete versions of languages that already exist. I wonder if the panel might have some words of solace at least for these people.

DOUG ROSS

The audience is full of shills. If it is possible to follow the kind of scheme that we are trying to promote--and again this ACM paper for this Washington conference goes into considerable detail on it--then the multitude of systems that we see as being necessary will all be processed by a single language processor working under a single operating system and as you go from language to language all you do is change the tables which control the behavior of these generalized language processors, so that--in fact, this is the only sensible way that I can see to solve it, because we must have different languages for these many different kinds of users and yet these people who run the computer installations also just can't face having 500 completely different master system tapes and all sorts of things to worry about. And so this is precisely what we have been trying to do is to find a middle ground where you have a single processor, there is a single way of making problem solving systems but it can be molded to be many different systems and in this way would be a very sensible solution to it.

MORT

Sounds like a user-dependent language.

DICK MANDELL

I think that Doug's answer is the right one to the question. I think there is one other aspect of it and that is that generally these compilers that are written to handle these special purpose languages are written themselves in a very, very concise language--or should be--and this language becomes in a way a documentation for the special purpose language so that you have a common language to talk about even though you're talking about all sorts of other things.

PARLAN SEMPLE

This subject of proliferation of languages is the number one concern I can assure you of both user organizations SHARE and GIS with which I've been working. They insisted that there be a solution, so we are definitely aware of the problem.

WEINDLING

I understand IBM has 168 languages?

PARLAN SEMPLE

I hate to keep talking about the subject I know best, but GIS, for example, considers it and I'm sure the other series of languages as being the forerunner of a new generation itself of software languages, so we have not only new languages but a new generation coming to or toward us.

PHIL DORN

That's GIS's theory. SHARE's theory is that it ought to go away. I think that would be a fair summary at least of the impression I got at the last SHARE meeting.

How do you read it, Dick?

DICK MANDELL

Well, I don't know if it's all "go away"--I think it's more we're going to go ahead and build. I'm not sure that there was any as things are that are not going to listen. Maybe we're not going to do anything, but they'll listen.

PHIL DORN

As far as Don Parker's original question on the language proliferation--it's quite clear that the way to get into trouble is to write your system in assembly language--you're looking for trouble. Yet the most common language we have--the language which I feel has the most commonality between implementation--and really diverse implementation--happens to be COBOL.

I find FORTRAN standards are very deceptive. They look very similar on the surface, but the darn programs don't run. We have had more luck with commonality between Cobol, surprisingly enough. Language standardization may, if it every becomes rigidly enforced, solve some of the problem. I think the best thing you can do is in your own installation limit the number of languages--and perhaps Doug Ross has the solution to that, at least for this area of computing throw away all the assembly language manuals or keep them under lock and key. It's a tough problem, Don. It's the installation manager's headache and I wish I knew the answer.

DOUG ROSS

We don't have a solution, but I think we're staggering in the right direction. How's that?

DICK MANDELL

I think the answer to the question turns out to be academic because although in the next year or two we will see a lot of false starts on the use of languages for design automation systems, I feel that PL 1 probably will become accepted by the other manufacturers, and will become the standard language. We see in this room that more than 60 percent of the people are using FORTRAN, which is not a suitable language, certainly not the best language but the best available under the circumstances. PL 1 is clearly better for this type of application and as soon as the conditions of acceptance and availability are met there will be a complete swing to PL 1 just as there was to FORTRAN.

MARVIN LING, GENERAL ELECTRIC COMPUTER (Phoenix)

I think I have comments on the need for languages for design automation which handles the mass data and also to describe information--in particular the relationship between the things you are describing and the property of things. Now in the lack of general acceptance of PL1 and the eight languages for computers other than IBM (such as General Electric computers). We are using FORTRAN and for this point I do not quite agree with your opinion as to using FORTRAN for design automation languages. We are up at Phoenix developing a project which write numbers of routines in machine languages and imbedded those routines into FORTRAN and consequently use FORTRAN to write the design automation problems. I think a typical example of this kind of system is . They are written by Joe and through our experience it worked rather satisfactorily under the circumstances that general design automation languages such as PL 1 or AED are available and assuming they are good.

PHIL DORN

Are you suggesting you use SLIP for this purpose or you've extended it to FORTRAN?

MARVIN LING

Yes, we have extended it in our FORTRAN by function primitives.

I suggest many of the people that are using FORTRAN for this application have done similarly--it's the only way it will work.

PHIL DORN

And therefore you don't have FORTRAN any more and when you go to the next machine you're back in the soup all over again. That's the trap.

NICK GARAFFA, RCA

I have a question to ask you--whether the dog wags the tail or the tail wags the dog. And I think the industry and users are looking to create a simile--they feel that the computer manufacturer is the dog at this particular time. Now a lot of the hands raised for FORTRAN IV were probably raised because we're going to lose our new found standards of 70-90's--these particular computers--and we were going to the new third generation computer. So a lot of directorates came out and said you have to get all your effort now in a machine independent language and FORTRAN IV was the only one. For a lot of people--the design effort had to go on, so you used FORTRAN IV. But that doesn't necessarily mean that this was the particular language that you wanted to use, or whether this was a good language, or whether PL 1 is a good language.

The question is that when you come up with a design automation problem you then pick an available language, and PL 1 looks like it would have everything that you might need--character manipulation, bit manipulation--but it doesn't necessarily say that this is the right one, two, or even for that matter, eight.

So I think the problem is going to be solved because when we finally decide what language is going to be the best one, lo and behold we're going to create another computer, probably in 73 or 74, and we'll be right back in the rat race again. So I don't really know if we're really attacking the problem. I think that none of these languages are directly applicable for the problem at hand. I think you have to make an analysis as to what you want to do and then you have to fit it into whatever language--be it AED or PL 1. You have to work around it.

DOUG ROSS

I think I sort of agree with the main tenor of the comment. It seems to me that the key thing on this business of people using systems because they're available and having to do with what's around, there is no way for the vast majority of people in the industry to escape from the fact that they are being paid for doing a job now. We are in the fortunate--or perhaps unfortunate--position of being paid for doing jobs in case they will be useful in the future, and I sort of like to think that these needs go hand in hand in that this is why we like to work very closely with industry and have feedback from people who try to use the techniques. But I also think the question of stability over various machine generations and even future machine designs hinges on this depth of understanding of what we're doing. It's all too easy to hide what it is you're really doing in the trappings of the language you're using at the moment. An I think that taking the easy way out, making these augmentations of languages by adding packages of procedures, basic functions and so forth, it's all too easy to put in a bunch of ad hoc things which really are just your present view of how you're quickly going to get to the solution and that you're going to trip up in the future as machines change and so forth if this is maintained.

On the other hand, if the things we all do can be gotten down closer to the problem's basic nature, then I think that you can't ask for a better ground for future stability. Anything else seems to be inconceivable that you could guess ahead of time at what the future computer is going to be or what the future languages are going to be--this we can't do. But what we can do is we can say we have today a certain collection of jobs that we're trying to do. There are two ways we can do them. One is with baling wire and scotch tape, etc. (and that's the way too many of them are done), and of course that falls apart. So if you can stretch a little bit toward getting more sound fundamental sort of scientific underpinning to what we all do, then that's the way to start getting a step up on this stability over the future generations.

PHIL DORN

Nick, I' d like to bring to your attention the history of the Bell Labs work on Snobal, which originally started out at Bell Labs--three fellows did it. It was written--if I remember right the first implementation was 70 90 only, then they expanded it to run on the 90, 94, 40, 44 or a direct couple. The last time I counted there were thirteen implementations under way on all sorts of machines, some conversational Snobol being done for the SDS 940, and just about every type of implementation you could conceive. The language has a very definite purpose and fits very nicely in the general scheme of things, and it' s going to be done and also it' s a computer.

If you do the job properly--whether AED-0 is the answer or not is what we' re probably debating here--but if AED is the answer and is done properly and you can describe the problem appropriately on machine X when machine Y comes along we who are users will be crawling over you who are vendors insisting that it' s part of your responsibility. We' ve passed the day where you can sell us a new computer with just FORTRAN, Cobol and directions on how to walk around the card reader to drop in more cards. We' re going to be asking for more things. Consider that a threat for the future.

Dick, AED is right--we' ll ask for it.

DICK MANDELL

I think it' s even more than that. You bring up Snobol, and I think Meta 5 is another example of languages that were simple enough for three or four people to program in a reasonable time. When we get to the point where we have very, very complex languages it takes a large number of people--a lot of man months or years to do it. And you' re not going to have it done by the user. On the other hand, you' re not going to have it done by the manufacturers with their current attitude.

I' ve just been on a buying trip and I keep saying why won' t you give me PL 1 and they say if you' ll show me three machines people won' t buy because it doesn' t have PL 1, we' ll do it. But we' re not going to do it otherwise, and I think it' s this attitude in the industry that is going to inhibit this kind of growth.

DOUG ROSS

I' d like to add one thing. This has to do with how do you specify or set up one of these high level languages and speak directly to the problem of transferring from one machine to another. I think if you merely write a big set of specifications for a language and then expect all the different computer manufacturers to make their own independent implementations to meet those specs, then you' re asking for a lot of trouble.

I agree this is the situation we' re in today. This is why the comment is perfectly valid that the true definition of the present PL 1 language is the IBM PL 1 compiler. This we can' t excape from. The compilers are the definition of the language. So it seems to me then the key thing is to make the compiler as machine independent as possible and instead of trying to transfer from machine to machine the specifications of the language, transfer the processor itself. This again is what we are trying to do with AED. We are bootstrapping it right now under the 1108 and the 360 and back to the 94 in the new compiler. But it's the same compiler. We're just just rewriting new compilers for these machines to meet the same language specs--we don't have the specs for the language.

Again, the compiler is the definition of the language. But notice the onus this puts on the implementation of the computer itself--that it must be as machine independent as you know how to make it. And here again if you look at most of the compilers, including all of the FORTRAN compilers, their main forte is that they take very careful advantage of the characteristics of the machine that they operate on. Well, you have to find some point in between where you can still have efficiency but have machine independence as well.

BILL ALLMAN, DuPONT

Perhaps there might be another approach to this problem. It may or may not be what you gentlemen are speaking to day, but from my personal experience I find that the most workable system is to divide the job up into three levels. At the top level is the user. To my mind the user is the fellow that has an engineering job to do. He has too much to do and not enough time to do it in. His interface should be with a production technician who understands his problem, who can understand the language and the communication means that he works with. This may even involve a field trip.

On the third level is the systems engineer who is concerned with--in our case we work with FORTRAN IV in an Exec II system on the 1108 which is very powerful. There are then two interfaces: one between the user and a production technician and the second between the production technician and the systems engineer. This is a very dynamic process. The top level there is no change whatsoever in what--in how he has the means of communication he has been using for years. At the bottom level there is continuous change as assemblers change, compilers change, problem definition changes, and things like that.

PHIL DORN

Just seems to me going around the problem the long way. You're essentially accepting classical techniques. The way to beat this kind of a game is get the user the facilities to do the job himself. He knows what the problem is. Describing it to anybody, particularly someone not at his professional level, is an extremely complex task. If you give the user the facility--call it a graphic console if you like--you've got the problem licked. I don't understand why you go to all these interface levels.

BILL ALLMAN

I think the idea is to get maximum utilization of the computer doing the most useful work for the functions of the organization. The users are very busy people. One of your purposes is to make more of his day than he has had previously--to make more of him as an engineer. We find it very worthwhile to have a specialist--a production specialist--who becomes very adept at formulating the problems in the program input form. He has capability of changing his work as necessary as the program changes. But his interface with the user is of a relatively constant nature.

By this way you have specialization and you avoid the problem of the multiplicity of languages and you avoid major changes in the way in which our engineers have to work--and you get to work on the box.

PHIL DORN

I personally don't find it the proper approach--I say why not change the way the engineers work. Maybe you have to sneak up on them a little bit at a time as described in the paper this morning from Boeing in Wichita. He sneaked up on them just a step at a time.

The thing that bothers me about your second statement was you were talking about maximum utilization of the computer facility--I'm much more concerned with the maximum utilization of the engineer's brain. The computer to me is the least of the problem. If you need more computer power you can always buy it. But I need that engineer's brain--he's a much scarcer commodity today than machine time.

DICK MANDELL

I think one of the major problems is that of communication. At least when you're communicating with a machine, if it doesn't understand it usually blows up, but when you try to communication with a technician or somebody that's doing the work for you, he'll do what he thinks you're doing and he'll do it the way he thinks you said, and maybe it'll be some time before he blows up because he's a little smarter than the machine. In this case I think it's unfortunate.

DOUG ROSS

I was sort of intrigued by the comments from the floor because exactly the same breakdown is the one I see for making the large number of languages possible. I think he was saying it made it so that engineers only had one kind of language. Maybe that's true within one company or within one group in a company, but take the many companies and the many groups and you're back in the many languages area. We do see a need for essentially the same three breakdowns coming down from the user, the end product language; there is a system programmer type who knows how to set up the tables that control these generalized processors so that you can then make the many different languages be processed by the same processor.

Then there's a third category which is the system programmer. These are the ones who take over when you go from one machine to another. They are the ones who bootstrap the general processor itself. So I do see the same breakdown of users and intermediate system setup level and then the--right down to the nuts and bolts the man who interfaces with the actual computer and operating system as a legitimate breakdown, but I see it in the light of making possible the many languages that are needed, in an orderly fashion.

PHIL DORN

Doug, that's the starting point of the game. After you've been through the job and got the user set up you don't want to see him until his problem changes or he needs something completely new. He functions on his own. And that, I think, is not what the gentleman from DuPont is describing.

DOUG ROSS

No, that's why I found it intriguing. I got exactly the opposite reading that he was getting out of what he was saying.

KELLY BROWN, McDONNELL DOUGLAS

I'd like to address this to Phil Dorn. Why do you have this favoritism toward Cobol? You could expand that just a little bit.

PHIL DORN

Good heavens! I'm the PL 1 Project Manager at SHARE, not the Cobol Project Manager. I have no particular favoritism toward Cobol, but if you give me a choice on any machine of Cobol versus FORTRAN I can do a heck of a lot more things with Cobol. And I could for a long, long time. The general implementations I have seen--Cobol is richer, it is awkward, your hand gets tired, the language doesn't understand subroutines--I can tell you all the things that are wrong with it. But it is a much richer language than FORTRAN. Let's face it.

KELLY BROWN

What you're saying is that if you had a choice between the two--FORTRAN and Cobol--you'd choose Cobol.

PHIL DORN

If you gave me one language to use in a general purpose computing shop, I will take Cobol. Because I can do things with Cobol that FORTRAN would never dream of, but I don't find the reverse case. If you just restrict me to those two. If you put me on a Burroughs 5500, then the game changes. There I have Cobol and Algol and a very much extended Algol. Then it's a different deal. I'm definitely not prejudiced toward Cobol--I wish it would just stop its development right there.

I'll put on my PL 1 hat for a minute. In Union Carbide we have had very little difficulty convincing commercial programmers to use PL 1. They find it much more comfortable. Our problem has been with the scientific programmers in FORTRAN.

CHARLES HOOTLINE, CAMBRIDGE UNIVERSITY

None of the panel members have mentioned microgenerators, and at Cambridge one of our graduate students, a chap called Peter Brown, has implemented a completely general purpose microgenerator called ML-1 which isn't oriented to any particular language but is rather just a straight processor. You describe your micro environment it it, take in one input string and outputs another string. We've had very considerable success with this general purpose microgenerator in implementing very quickly and readily special purpose languages.

In particular we have used it to implement a systems writing language and then having decided what features we wanted in our language, using this general purpose microgenerator, we are now producing a compiler for this language, actually using the Brooker-Morris compiler on our Atlas computer.

This microgenerator itself is implemented on our PDP-7 and on our Atlas computer and it is in fact written in terms of micros which is a part of the language itself, so when you transfer it from one machine to another one only has to recode the few very basic micros, and this is in fact how it was transferred from the PDP where it was originally written onto the Atlas computer. Now during the summer it is going to be put on the 1108 at the National Engineering Laboratory in Scotland and also on some ICT range of machines. I believe IBM did look into this in England but they decided, I believe, not to use this now.

I'd be most interested to hear the panel's comments on microgenerators because I feel they really are an essential part of any general purpose language system.

DICK MANDELL

I think the intent of Meta 5 and the intent of Doug's AED is to do much the same thing in the case of Meta 5; you describe the language you're talking about in terms of a microgenerator package and I believe in Doug's case you talk in terms of a table that you put into the machine. And the intent is certainly just to be able to build languages in somewhat the same fashion you described here except you're not tied to an assembly language--it's possible you don't need an assembly language. And also in PL 1 there are the compile time features which do something--not too good, but they do some of this work.

DOUG ROSS

Well, as Terry well knows we do have a micro package in AED-0 and we think of it as quite an important part of the language. Coming back to the viewpoint which again is expressed in this ACM paper, there is a definite place in the generalized language processor that we like to think is the way to go about things for micro pre-processing. We think there are four major phases--the lexical phase is the first one; that's where you group your characters into multi-character items or syllables of the language, the parsing phase which is where you do the structuring of statements made in the language, the modeling phase which is where you make a representation of the understanding of the problem, and the analysis stage which is where you actually end up solving the problem. Now between the lexical phase and the parsing phase, right at the very beginning, is the place where we see micro preprocessing as being appropriate.

In other words, we don't like to think as a general scheme as being that you take all of the last phases, the parsing, modeling and analysis stages, and reduce them to just what an assembler will do. We'd much prefer to think of the total scheme as having essentially five phases, then, with the micro phase properly being an item-to-item rather than character-to-character manipulation, and what a micro pre-processing phase does is it allows you easily to augment a language without actually extending it, i.e., to put in different forms for the user to write without actually adding more things to the semantics of the language. I think the use of micros in the way that Charlie was outlining is possible only when you stay within the semantics of the target character string--you're going from one character string to another character string and the final character string is in some language. So you're restricted to the semantics of that language. Of course, if that's a machine language assembly language then this is essentially no restriction because you're going to a Turing machine sort of, but I think the proper role for micros is not to try to take over the whole job but to play this very crucial in-between job between what the user actually writes and what the language processing the semantics and modeling and analysis phases are actually going to do.

GLOCKING, ENGLISH ELECTRIC COMPUTERS

I havetwocomments about this discussion. One is it's my opinion that languages don't really matter. It's the storage system and your data handling facilities. The multitude of your programming in design automation is dealing with multiplication of level control for production purposes. If in fact you can deal with this with some kind of a general information handling system you've beaten at least 90 percent of the problem. I think Doug Ross is saying exactly this: that that is the important part of the system. To this end, a paper this morning--the Naval Research paper--had a very interesting way of organizing files.

The second thing is that this name Meta 5 is peculiar. They're trying to standardize on languages by using some kind of Meta language. But Meta 5 is presumably the fifth version of this Meta language. What happens when Meta 6 comes along?

DICK MANDELL

There are, as I'm sure many of you know, a number of somethings we call Meta compilers running round the country. In Los Angeles there was a development group started some years ago and Val Shoury wrote a language called Meta 1 on the 1401 which he promptly used to generate Meta 2 on the 1401, and these things got spun around for a number of years. Last year I put a paper in the proceedings of this conference on Meta 3 which is the language I was working on 'till then and, unbeknownst to me, there was a Meta 5 developing at SDC which apparently realized that Meta 3 was around so they picked another number. Meta 4--somebody reserved the name since he felt he liked metaphor and that should be a very excellent name for a Meta compiler. So that never got written yet.

At any rate, Meta 3 was going on toward Meta 5, and I was really quite pleased to see that I was going in the same direction that somebody else had though of, but they got there a little ahead of me. That's why Meta 5. There's also an M. E. T. A. which is being developed at SDC which is a language which operates on trees. Our intention is to also put this tree climbing kind of thing on Meta 5 and we may well decide to call it Meta 6 just because it's different.

TOM GRUBER, RELIANCE ELECTRIC CO.

I'm just looking for perhaps comments on languages or techniques for capturing logic, and I'm thinking in terms of things such as decision table compilers. I know at one time--and I believe there is still activity in this area. I wonder if anyone from the panel would care to comment on this.

PARLAN SEMPLE

I hate to ask an embarrassing question, but were you at the session this afternoon? (Response - no, I wasn't.) I tried to illustrate there taking a decision table and treating it as though it were a computer file. Although I took a very simple example I did not get into the hierarchic structure which is available with MTA because I took a very simple, one record level format and I believe I showed in the paper that I could manipulate this compiler as required using the procedural logic inherent in GIS.

One further comment here, although GIS is not a microgenerator, I hope most of you realize that it is, in addition to being a language which we are addressing here today, it is backed up by tremendous programming systems as a result of
hoped to be, but in any case it's a programming system which generates codes. We can go code as a result of the user's statements. And if the user's statements define decision tables, fine, we'll process that; if it defines something else, we'll process that. The generalized system--we care not what the English looks like.

DOUG ROSS

Just a point of information for you, although I can't remember the number or the status exactly, there is enough interest in decision tables and languages for them for the U. S. A. S. I. Standards Institute to have just recently set about setting up a standardization activity for decision tables. I could look up the information if you like if you'll write me a letter about it.

PHIL DORN

At the same time, Doug, I think there's a decision table processor for 360 available. Seems to me somebody at North American recoded Detab X in PL 1 to run on the 360, I think. There's still a smattering of activity in it--not a great deal. A good deal of it is concentrated here in Los Angeles and a couple of firms in the east are very active in decision tables--McGraw Hill is one--just here and there in spots around the country. But there is a decision table processor for 360. At least one, and maybe two. Richfield Oil I think might be a good contact for you.

MARK MAIDEL, DOUGLAS

I have a comment to the man who asked the first question about what will the computer managers do. I' d like to throw back this challenge to him by saying that most computing that is done is essentially an analysis of special cases, and that somebody has to decide what is to be done when each of these special cases occurs. The output of this decision usually is a bunch of coding, whether it' s in assembly language or FORTRAN or any other language, that very seldom is any documentation made of what special case he is tested for and what action he intends to do it.

Now, in a production firm, typically they cannot afford to allow a skilled man to sit down and spend time documenting what he has done, and also if this had been done and released to competitors, the competitors would not have to spend the same time and money to generate the same information. However, if the documentation was done, then the conversion from one machine to another would, I think, be rather trivial because the hard part has already been done.

Now the second comment that I' d like to throw is to the people in general who design computer programs who are going to aid someone else, and that is please stop taking things away from him when you try to give him something. I cut my teeth on the 650 and this had a very beautiful little instruction that' s called the Shift and count. The 90 and 94 came along and we lost that instruction, and all the problems that I have solved using this particular technique are gone. I have to write my own macro. When I communicate a problem and can solve it in FORTRAN, I try to do this because our company is now saying "solve problems in FORTRAN." Now many times I make use of nonstandard features just because I' m under this restriction. But if I can solve a problem in FORTRAN or in some assembly language, let me do it because presumably the problem that I' m really trying to solve is the one that you don' t know how to solve either. Or else I wouldn' t be in the loop interacting with the computer, but rather someone else could have filled out the load sheet and the problem solved in the batch process and we wouldn' t need graphics or on-line... Please stop taking things away from us.

NOLAN DEHN, McDONNELL DOUGLAS

I was just wondering and I would like to put the thought to the panel as a matter of record for this discussion as long as it is being taped, and perhaps as a contribution to helping Doug Ross and his thought of coming up with a system for generating what we need. Is it known among the people involved the requirements--what is different about the design automation activity? What are the requirements, then, for a "best" or good programming language? Would this be beneficial at least to get on record?

DOUG ROSS

I would just again refer to the paper for this fall ACM meeting in which we don' t try to say what' s needed but rather to provide a framework within which to do the job. I don' t think you can crystal ball too much more.

ALAN STONE

Clearly we could write a specification for design automation programming language, and I think we all agree here that we don't want to and shouldn't, since there are existing languages that come close enough to do the job it's not required to invent another language. I think I sense a general agreement among the panel and most of the audience that PL 1 is suitable but its acceptance now is limited by its availability.

Another comment on some of the things we've talked about--the Meta compilers, natural processors and even the source language micro capabilities that are in PL-1 now or eventually will be in PL 1 and similar languages all are useful tools to develop user problem oriented languages. I don't think many of us are willing to go out on a limb and use those tools to design our own design automation development language and thereby picking up responsibility ourselves for transferring that from machine to machine.

The gentleman mentioned that the micro processor, but my experience with these devices is that the job of rewriting the primitives while a finite task certainly much simpler than rewriting all of your applications programs is still a formidable job. I think many of the people developing design automation systems are not willing to take upon their own shoulders that of designing and maintaining and transferring from machine to machine their own compiling system.

PHIL DORN

This job of transferring micros from one machine to another--the classic McIlroy string package implemented originally on the 90 to my knowledge has not yet been implemented on the 360 and I don't think they're going to do it. The assembler just won't support the things that McIlroy does in there. So its not always that easy going from one machine to the other at the micro level. I don't agree with the other generalized approaches 100 percent--it would be against principle. But I think he's getting closer to being able to define the problem than anyone else has yet come. Maybe after his next paper we'll change our minds about that.

DICK MANDELL

I think I'd like to reiterate what Phil said about IBM being very interested in ideas at the SHARE Field One Project meeting. They were crying for ideas or things in PL 1 that are not already in PL 1. They weren't terribly interested in changing things that already were in PL 1, but they were wide open for suggestions on things that weren't, and I think it's a powerful offer they've made to ask for these things and that there is interest in developing these things and it can be done.

JOCK RADER, HUGHES AIRCRAFT COMPANY

I'm not sure I understand the purpose of this language, but it seems to me certain people are suggesting that the user himself would be writing this generalized language--the engineer himself--and if this is true, and in my experience the design automation programs I'm familiar with run into thousands of statements and require months to develop and debut, and then additional months perhaps to alter them. I'm wondering if now the engineer is going to be shouldered with this and if his time is so important, is there going to be design automation as I get it.

One of the things I'm trying to ask is--is the purpose of design automation then going to be design of specialized language which the user himself can use, and this is all he does? Or is he actually going to be sitting there writing a program in conjunction with the user--the engineer who is actually getting the results that he then bases his answers on?

ALAN STONE

I don't think we're talking about user developing design automation programs in currently available languages--we're talking about special purpose, problem oriented languages for specific tasks that a user--an engineer could use if it's suited to his problem it's easy to learn and it accomplishes his job. He needs very little training, very little help--this whole discussion has been on two levels: the design automation languages for the user and design automation languages for implementing design automation systems.

JOCK RADER

Okay, so you would actually have a combination then. For some applications you'd have a language which the user himself could use directly and in other cases you wouldhave an intermediary who is actually running--

ALAN STONE

However, the most of the languages for implementation and one of the requirements for it would be that both the language and the system for implementing design automation systems--one of the requirements would be that it be suitable for generating higher level, problem oriented languages.

PHIL DORN

The user level--the guy who is really trying to get the work done--is not the same level as the system building level. If in fact we build a system and we can't produce a language suited to this particular discipline easily and quickly, we're in big trouble. The disciplines using the computer are changing and expanding rapidly. I should be able to produce a user language for a psychologist, sociologist, librarian--it's essentially the same problem. But I better be able to do it quickly if I'm going to have that many of them around. But I don't see the user building the compiler. I would greatly value his assistance in the language that he needs, because I don't know his problem and I've got to have the tools or I'm in trouble.

DICK MANDELL

I think that these special purpose language compilers really stand just as input processors to the type of design automation programs that you're building today, only maybe bigger ones. So the design automation man simply takes on an additional task of building a processor that makes it a little easier to communicate. But I don't think his role has changed.