RECORD OF SOCIETY OF ACTUARIES
1994 VOL. 20 NO. 3B

COMPUTER LANGUAGES

Moderator: L. TIMOTHY GILES
Panelists: MARK A. CAVAZOS
KENNETH KOFFLER*
FLOYD R. MARTIN
DENNIS R. TOBLEMAN
Recorder: L. TIMOTHY GILES

This session will allow you to vent views and prejudices regarding the advantages and disadvantages of using several computer languages for actuarial work. Normally the languages we have experience with are the ones we feel are the best. But do we really know the strengths and weaknesses of languages we haven't used much and don't really know? And when should spreadsheets be used?

MR. L. TIMOTHY GILES: Meeting at 8:00 a.m. on the subject of computer languages is like drinking bourbon before breakfast. This is an interactive forum; an actively moderated session with significant audience participation. I learned *A Programming Language* (*APL*) about 20 years ago, and I learned spreadsheets about ten years ago. I am defensive about *APL*. I don't want to learn anything else. Recently our company decided to get a new life insurance administration system to move off the mainframe and onto the PCs that we all had. I was part of the investigating group and it was awful. We interviewed about four companies and two of them kind of stood out. One had a server, I didn't know if that was good or bad, and it had a modern, fourth-generation language. I don't even remember the name of it, but I do remember it couldn't do square roots and we just dropped that one. But another one was written in *Magic*, and the developers bragged about it. I brought this magazine, *Insurance & Technology*, which has this big ad on the back cover that attacks *Common Business-Oriented Language* (*COBOL*) and advocates *Magic*. Now Magic, that's a company, says it was the winner of the international developers competition for the second year in a row. Magic had a total of four winning teams in the top 15, more than any other product. *COBOL*'s top finisher was a distant 40. I don't know anything about that competition. Maybe it's like a county fair where everybody's tomato gets a ribbon. But I think that's the hook; that got my attention.

Now I'm going to introduce our panel, and they're each going to stake out a position and defend a language. They'll do that in about five minutes, and then we hope that there are some of you out there who will defend a language. We're not going to cover all the languages so we hope that some of you will stick up for some of the others. The people who designed this session wanted it to be kind of confrontational. Floyd (Ray) Martin is a health actuary with Tillinghast in St. Louis. He's going to talk about *FORTRAN*. Dennis Tobleman is a health actuary with American National. Mark Cavazos is a pension actuary from Dallas. Ken Koffler is here from Magic, Inc. Ken is not an actuary, but he's going to talk about fourth-generation languages in general and about his own language in particular.

---

*Mr. Koffler, not a member of the sponsoring organizations, is a Systems Engineer at Magic Software in Dallas, TX.

MR. FLOYD R. MARTIN: I'm going to talk a little bit about my experience with *FORTRAN*. Some people still do programming in *FORTRAN*. Whenever I mention that, some people are surprised to hear that *FORTRAN* is still alive and well. My first experience with programming and computers began between my junior and senior year in high school. I was at an academic camp that summer, and we had to do some programming. The language they had available for us was *FORTRAN*. We had to use the old punch card machines. Does anybody remember those punch cards? An operator would feed them through and run your program. You'd usually get about ten pages of syntax error sheets after submitting and then you would have to do it again. But I found it very enjoyable and I found *FORTRAN* to be fairly logical and work in a logical manner. When I was in college, I then had some courses in computer science, and I was exposed to some higher-level languages. I think one was called *Lisp*, which is a word-arrangement-type language. However, for most applications, I found *FORTRAN* to be very useful and very manageable for the things I needed to do.

I used *FORTRAN* to do some graphics. I actually plotted graphs of the world. I made models of the *Enterprise* using *FORTRAN*. I also did some statistical studies of Monte Carlo simulations; that was my first taste with *FORTRAN*. It seemed to me in college that *FORTRAN* was one of the more accessible languages at that time. It might be the era that I'm from, but I found it to be very accessible on the college mainframes.

Then when I started in my first actuarial position, *FORTRAN* was available on a time-sharing system through an external mainframe. The company had its own mainframe, but it still ended up buying services from an external mainframe for doing a lot of the actuarial work. Back then, you still had to do line-by-line entry in *FORTRAN*. You entered a line, you got an output line. You entered information, you got an output line. It kind of kept you moving up on the paper. Before they had time-sharing on monitors, this was all done on paper so you accumulated quite a bit of paper. When we later got a PC in that office, we were able to do more of a rolling on the screen for data entry and so forth. But it was still the line-by-line-type data entry so you just kept scrolling.

In my current position, *FORTRAN* was available again on the mainframe, and later the company acquired PCs and I began using *FORTRAN* on PCs again. There were some subroutines and functions made available that allowed for menu-type entries and some screen manipulations, so you no longer had this scrolling data entry or output. You could now create menus on the screen, select items out of the menu, and get reports up that would start at the top of the screen and fill the whole screen. But when they came out with the VGA monitors, they weren't supported by these subroutines and I almost had to learn another language. I started looking into *C*, and I even got a translator that would translate my *FORTRAN* programs into *C*. I just about went over the edge and then *FORTRAN* came out with some new software that had many neat graphic capabilities so I was able to bypass that problem. It would support the VGA monitors, which then helped a lot because everybody was going to VGA, and I needed to have that capability in the programs that I was writing.

I considered learning again other languages, but the time involved, the learning curve involved, has been prohibitive when I take into account other things I could be doing, both with the work I'm at or even spending my own personal time. So as long as *FORTRAN* has done the things that I've needed, I have not felt the need to spend the time to learn another language. Although it would be interesting, it just hasn't been a priority for me. I think when you learn a new language, you need to have a constant usage of it, you need to be exposed to it daily. I would even probably have to take a college-type course, and that would require spending a lot of dedicated time working with that language. If you're learning a new language, you have to spend a lot of time with it to become proficient. As time went on, more of my work though has gone toward using spreadsheets on a day-to-day basis. I use *FORTRAN* for more complex data manipulation, things that would take quite a bit of work to develop on a spreadsheet that I could quickly write out in *FORTRAN*, such as reading in data and spitting out the appropriate numbers. But for most of my day-to-day number crunching, I use the spreadsheet.

In conclusion, I think that *FORTRAN* is still a very viable, very powerful language. I think it has been enhanced a lot from the old original *FORTRAN* syntax. There are many new functions and new subroutines available for using *FORTRAN*. A compiler is specifically dedicated to the 32 bit, so it will not work on lower machines. Microsoft is still supporting *FORTRAN*, and I think that it's still a very useful language. The time involved for me to go and learn another programming language would just be prohibitive. Again, the spreadsheets have become very useful to me, and on a day-to-day basis, I use spreadsheets almost entirely. I use *FORTRAN* for major projects and things that would not be appropriate for a spreadsheet environment.

MR. DENNIS R. TOBLEMAN: I've used *APL* for about 20 years. I've used it under mainframe environments on CMS, OS, and MVS. More recently, I've used it almost exclusively in a PC environment. The PC-*APL* tends to be superior, in my opinion, because the company that supports it has done a lot more to produce many peripheral and auxiliary programs. Of course, it's its main stock in trade. IBM-*APL* is the other mainframe *APL* nowadays. They never have developed it to the same extent.

I'd like to tell you a few of the things that I think are *APL*'s particular talents. First, as an actuary, we tend to like things expressed in mathematical constructs. The symbols and the program syntax resemble math equations, if you can overcome the fact that *APL* does execute from right to left. That's something that causes some people many problems. Second is the ability to do vector and matrix manipulation. If I'm seeing a page of rates, it's nice to be able to think in terms of working with a page of rates or a number of pages, a vector at a time, plan-specific for instance. Nested arrays is something that I haven't used a lot personally, but it gives you the ability to have unequal subsets of data within a common variable. It's a very nice feature, very unusual.

The file access is good in *APL*. I'm talking primarily now about the PC environment *APL*, but in particular, the thing I like is the ability to create files with keyed index and being able to say, I want the record with this key. I've always liked that, even though it's a little bit inefficient and time-consuming. I think the most important consideration is man-hour time and productivity. And as the power of the PC continues to advance, it covers up a lot of the sins for my bad coding.

Next and something that is very important and specific to *APL* is the fact that it is an interactive language. That means several things, but sometimes I like to get into the *APL* environment and just kind of play around with numbers and use it kind of in a sophisticated calculator mode.

Second, if you do write a program and you run it and have an error, it will stop at wherever it encountered the error. It will freeze right there and point to the place where you have the problem. It will give you the option of trying to fix things, because you can see the value of all the temporary variables that have been created up to that point in time. You can fix it and go on, and that's a very nice feature.

Third is something I think that people tend to underuse a lot. I know a lot of you are good *APL* programmers, but that's the stop function capability. Often I'm not sure if my own program is doing what I want. The stop function stops on any given line that you choose. You can just play around with the variables, the arrays, etc., that have been created up through that point in time. It helps keep you from racking your brain saying, well now, if I do this outside matrix multiplication, for instance, or whatever, am I going to have the rows and the columns in the right format or not?

Finally something that's fairly new to *APL* is its WINDOWS product. It takes advantage of the 386 and 486 chips to give you a little extra processing speed. I've had only a few weeks to use it. Of course you all know what WINDOWS compatibility means. Believe me, it's a very nice thing to work in *APL*, send it down and call in your spreadsheet. You can now interchange more freely between spreadsheet and *APL*, and a couple of spreadsheets are supported. I personally like doing calculations in *APL*, I maybe spend half the time getting the program to do what I want and then the other half getting it to print out what I have. Maybe I'm too particular with my headers and the like, but the ability to export your data variables to a spreadsheet environment is very nice. Also, I think we need to think in terms of the fact that we don't want to have actuaries responsible for all the data input. So you can get people who are familiar with spreadsheet to create data input that you've designed. Then you can create a bit mapping and bring them into your *APL* work space and do your calculations there. Well, those are some of the advantages of *APL*.

MR. MARK A. CAVAZOS: I'm here as a user. Unlike these gentlemen, I'm not into programming languages like *APL* or *Magic* or any of that. In fact, I started with computers when I was in college. I played computer games, and I learned enough *BASIC* or *APL* to go into the computer program. I manipulated it to do other things, and that's what I've done. I worked with Mercer for 13 years, and now I have just started going out on my own. That's what I've done in my career. I've taken the languages, the software that has been developed, and I have manipulated them either to find some techniques that were built into the system that no one ever uses just to see if I could get the system to do something. Or I will take a software that's been developed and get it to do things for which it wasn't originally programmed. I've also taken the data that have come out of software and used them like a spreadsheet because the particular situation I was working on was not available with the software that we had. So now I could look at it as many of you do, trying to figure out what kind of language would be useful and how the whole process would integrate from getting in the data to actually printing out a report and any supplementary items after

that. So I may end up actually having as many questions as the audience for the other panelists.

MR. KENNETH KOFFLER: What is *Magic*? *Magic* is a post-fourth-generation, object-oriented, code-free, application development and deployment tool. *Magic* allows you to develop applications quicker than any other tool and more importantly, allows you to make changes as the business requires. Consider *COBOL*. *COBOL* used to go into your favorite text editor; it used to write 500,000 lines of code. Now maybe you go up to a fourth-generation language (4GL), which now is at 25,000 lines of code. Now in *Magic*, which is a post-fourth-generation language, instead of writing code, you describe what you want to do in tables, and now it's only 500 lines. So which is easier to maintain, 500,000 lines or 500 lines? If you as users want to make changes, is it easier to do it to a 500,000-line or a 500-line program? That's how *Magic* gives you the ability to develop applications quickly. And more important is look and feel. In the olden days, on most tools you'd have to have it look some way based on what the tool required. With *Magic*, you can go in and have any look and feel that you want. You can use colors, or buttons, or pop-up or pull-down menus, or check boxes allowing your users to be very, very productive. They get the look and feel exactly as they want it, which allows them to develop and see their applications and make it fun to use.

Now let's talk about how *Magic* does this. In *Magic* you go into a table and you describe what you want to do. There are 13 operations that will allow you, once you learn them, to write mission-critical applications. *Magic* uses object-oriented features so I can make a change at the very highest level that will convert all the way through the whole system.

Now let's say we developed this application. It's sitting out here, and now for most systems, you're stuck on a particular hardware platform or in a particular database. *Magic* allows you to have independence. I can work on the weekend on my DOS machine at home using B-Trieve. Let's say I go into work and I have a VAX or a UNIX machine. I can instantly port this application to this other machine and convert it to a different database. I have a screen here. On the screen, I have some information, maybe from DOS, or B-trieve, or any of my X bases, such as dBase or FoxPro or Clipper or whatever; or maybe information from my UNIX machine, such as a Sun and Sybase or Oracle or Informix or CIM; or maybe from my VAX, my Digital Equipment in RDB or RMS, all sitting on the same screen, with *Magic* handling all the database manipulation or cross platform, cross database for you. And maybe I have some users sitting there in DOS and others wanting to run this application under WINDOWS, the same application without any changes. I have some users running native on the terminal and all are looking at exactly the same application. *Magic* is handling it cross platform, cross database. If you look at a client server, *Magic* allows you to split the data, the application logic, and the presentation anywhere you want, wherever makes the best sense.

So what does *Magic* give you in the end? It really gives you productivity. It lets you develop applications quicker than any other tool, and it allows you to make changes at the highest level and aptly go and convert it through the whole system. It gives you platform and hardware and database independence so you can move this application or put it wherever it most belongs.

MR. GILES: I want to ask the audience, have any of you ever heard of *Magic*? (Nobody's ever heard of it.)

MR. KOFFLER: Let me also add, *Magic* in the U.S. is not as well known. It's actually written in 18 different languages, and it has 250,000 users. It's the number-one tool in Japan and Europe. It just came into the United States in 1991, so it's a little new here.

MR. GILES: I want somebody in the audience to defend or advocate *BASIC*.

MR. RICHARD E. ROWAN: It's cheap.

MR. KOFFLER: Let's talk about this for a second. It definitely is cheap to buy that software, but then you're talking about your programmers' time. All of a sudden you have a project with 100 programmers that five programmers could have done faster, quicker, and easier. So in the long run, which is actually cheaper? You can look at the long-run cost of a software and the many years of use. Yes, it might cost less to begin with, but in the long run, which is actually going to be cheaper and which is going to be easier?

MR. GILES: One problem many of us have is that it's not our choice. The company or the firm is picking the language, and we're kind of tagging along behind, I realize that. I want to get other languages represented. How about *C*? That's a popular one. Does anybody use *C*, or *C plus*, or one of those types?

MR. W. KEITH SLOAN: I don't personally use *C*, but several of the people in our shop do, and they use Clarion Development System, which is very similar, I gather, to the *Magic* that's been demonstrated.

MR. GILES: OK. Another one is *Pascal*. I don't know any of these, but how about any other language?

MR. EDWARD G. BAILEY: We often use SASS in our shop. It's a nice package. I don't know about the rest of your programming needs, but most of what I do is write a program and not use it again for another year. It's not meant to be used repeatedly. And so I want something I can code quickly, make it work right, and not have to deal with it again. And if I do, it's simple enough to read that somebody else can pick it up, read it, and make the changes to it next time.

MR. GILES: What type of actuarial work do you do? Are you health or pension?

MR. BAILEY: LTD.

MS. JEAN M. WODARCZYK: I am one of those actuaries who doesn't go beyond a word processing package or LOTUS, but I have the responsibility for deciding the kind of languages our staff should use. We currently use SASS to do our health care data work. Perhaps that's a little old; I have the feeling we're not on the leading edge where we need to be. We also have been looking at some material that's written in *APL*. I'm concerned about getting involved in *APL*, because it's been around a long, long time and I don't think it's in a WINDOWS environment or is about to be. I'm

wondering if that's the right avenue to take, or if there are some other languages we should be looking at in today's environment. So that's kind of where I am.

MR. GILES: Dennis, I believe you said *APL* is available in WINDOWS.

MR. TOBLEMAN: Yes, it is. *APL*'s strength obviously is in hard calculations. It's primarily an actuarial and engineering tool. If people want to do a lot of report writing from databases or whatever, created from other means, *APL* is not going to be that good. It's a little complicated. You must make sure that you have backup, so a certain number of people must learn it. It's not the easiest language to learn; I expressed that in terms of actuaries. It's very natural to what we do. We deal with formulas, we see equations, that's our bread and butter, so it doesn't throw us to have to look at all that. But I understand there are different kinds of smarts in this world; many people smarter than I have trouble with equations. So I would first try to narrow it down to the type of applications you're going to be looking at. Are you going to be doing many calculations, etc.? If you are, then it may be a good thing. With the WINDOWS compatibility, you can call up and say, "Oh, shucks, I forgot to send out this memo," so you can jump over to something else. It does have greater export-import capabilities, which make it easier to take advantage of *APL*'s strengths without being burdened by its weaknesses.

MR. GILES: I have never heard of SASS until this meeting, and I've heard it a couple of times now. How many have heard of that? Most of you—wow!

MR. GREGG E. LITTLEFIELD: SASS seems to be used a lot at the university level. I suspect that's where most people saw it and that's where I did. Many universities have it, but that's not what we use. I was just going to mention another thing if you're not in the hard calculation mode. One of our pension actuaries recently fell in love with Microsoft Access database. He was keeping track of the valuation data and what had been done and whatnot. He's doing Microsoft Access now, and he likes that quite a bit. It's a relational database with a lot of WINDOWS interfaces, and what he had done in *APL* or had been doing in other, more primitive databases, he has actually moved to Access. So that's another tool that's out there. Some of these databases often are very sophisticated, and some of the things you can do are quite amazing. And so that's another thing we can think about here.

MR. GILES: Is Access a type of spreadsheet?

MR. KOFFLER: It's actually a 4GL development tool. It's a very good development tool. It's very object oriented, to be able to touch and pick. It's more of an end-user tool. When we're describing tools, we probably should place where they fit in the global scope. Are they a development tool? Are they an enterprise line of tool? What is your company looking for? For example, Access is a very good, very object-oriented tool, but it's more for development. But it makes you develop things very quickly and let's you do some things that are very amazing.

MR. GILES: What is a relational database?

MR. KOFFLER: I'm not sure I can give a good description of this that people are going to agree upon. A relational database basically allows you to access your data.

A flat file has, like B-trieve, a file structure with one file record after another record sequentially. A relational database has your information so you can instantly get to any of your data. You have a back-end engine, a relational engine, that helps you do this. And it does it by writing sequel commands to the system. I don't know if that gives it or not.

MR. GILES: I can tell you that from the life insurance end, they like these relational databases. If you have five policies and you change the address or the beneficiary on one of them, they can do it all. You don't have to go find the other four; they're all related and that's a big deal.

MR. MARK F. HOWLAND: Like Dennis, I grew up using *APL* but when I came to Blue Cross, I found SASS in wide use, and that's what my entire shop uses. It's also used by other departments besides actuarial: our provider department, our market research area. So it's a program that can be used throughout the company. It does require a little bit of an investment in training. It's not readily usable but after a three- or four-day course, you're as good as anyone else.

But what I wanted to also mention and ask others to comment on is support. I think the issue of support is very important and the SASS Institute is very, very responsive to questions and problems. It also runs national and regional user groups. It has bulletin boards and so forth so that you can share ideas with other users and with the SASS Institute. We found that to be very important and very helpful. It also runs on the mainframe and on the PC, and you can even run it in a production mode. We've worked with our systems people to actually run some SASS jobs in production as well as on the fly. So I think those are more advantages, but I'd like to hear comments about support for the other languages.

MR. GILES: OK, let's have the whole panel run through that. Ray, how do you get support for *FORTRAN*?

MR. MARTIN: I use the Microsoft version of *FORTRAN*, and if you've ever tried to get to Microsoft for anything, you know it takes a while. Usually, trying to explain a programming problem over the phone is very difficult. I would say Microsoft usually will get back to you, they usually will help you, but they don't always give you a good answer. But Microsoft is a big producer and it's hard to get through sometimes. If at all possible, I don't try to access them, I try to figure it out myself.

MR. GILES: Dennis, how about *APL*? When you have an *APL* question, what do you do?

MR. TOBLEMAN: Well, a few times I had to get support from IBM concerning its *APL* product, and of course it did respond, but it wasn't necessarily very quick. *APL* is not a big product for IBM. I've always found the support of the Manugistics Corporation, which used to be Scientific Time Sharing Corporation, to be quite good. It's not necessarily instantaneous anymore. Sometimes the company will take your name and your number and they'll call you back. The people they have on the help line to help you with problems are sharp. Nowadays though, you do usually have to pay a little extra. I think you get a 30-day support if you buy one of the *APL*

packages, and I think you pay a little extra if you want extended support. It's not that costly, but if you purchase that, I've always thought it was very good.

MR. GILES: OK. Ken, does *Magic* support its users?

MR. KOFFLER: Actually, it does. It has 7 a.m.–7 p.m. free support, it comes with your maintenance. It also has a bulletin board, it's on CompuServe, and it has user groups throughout the country. It has about 16 training centers throughout the U.S. too.

FROM THE FLOOR: Do you maintain these tables and then *Magic* produces the code?

MR. KOFFLER: That's a good question. There are seven tables that relate to each other but they only create one file. This file is a file you give to your end user, so that allows *Magic* to be database independent. So whatever system you're working on or whatever database you want the application to sit in is the database it will use.

FROM THE FLOOR: Well, does it produce code in some language like *COBOL*?

MR. KOFFLER: It actually has a *C* engine, a compiled *C*-executable that sits on whatever system you're using. And this one file is being interpreted by the engine. But 98% is the *C* engine, the other 2% includes the seven tables that create the file that are being interpreted by the engine. Does that make sense?

FROM THE FLOOR: Yes.

MR. MARTIN: So *Magic* is an interpretive language, it's not a compiled—

MR. KOFFLER: Well, actually 98% of it is a compiled executable. The other 2% are, basically the parameter is telling the engine what to do at run time so it allows it to be very dynamic.

MR. TOBLEMAN: Could you explain just a tad about these seven tables, because I'm still having a hard time understanding it?

MR. KOFFLER: Sure. This is sort of hard to visualize without a system, but first think about having a tight table. This is a global table sitting above all my files and programs in which I cap all my basic primers. Maybe I have a phone number set. I can set it up one time, and then I would set up files. In these files, I'd say I want to use a phone number, and I'd already have it defined. So instantly I allow my development to have one set so it allows easy maintenance and also allows everybody to be very consistent. Because if I want to make a change, I will go to the very top table, make a change, and it will automatically go through all the lower tables. And then, of course, there are some more object-oriented features you can do with the high level. Then the second level includes your data structures, your files. *Magic* will automatically go to a relational database and grab the structures for you. Or if you're using a nonrelational database, you would describe what your files look like. The third level is programmed; that's where you're actually writing your programs and

you're describing what you want to do. And then we have security and pop-up and pull-down menus and so forth.

MR. GILES: We've heard the terms *interpretive language* and *compiled language*, and many developers use that as a point of debate. What are those and what are the pros and cons? I guess Ray actually used the term; would you like to be the definer?

MR. MARTIN: Well, a compiled language is one that produces final source code that runs directly on the computer without any other external programs. *LOTUS* is an interpretive language; I would consider *APL* an interpretive language; there's a program that's running your program. A compiled program will run much quicker than an interpretive program.

MR. GILES: What is SASS, interpretive or compiled?

MR. MARTIN: It depends.

MS. DEBRA K. TOOHILL: A compiled language takes your code, translates it into machine language, and runs directly from machine language; it runs faster. It is harder to debug because you get these mountains of syntax errors. I wish I knew what happened with *FORTRAN* since the 1960s. I'm now an *APL* programmer, and you can't get me away from *APL*. I like it for what I do. But I was interested in learning about *C*, and I'm kind of disappointed that there isn't somebody here to tell me about *C*.

I was writing a program and had an error that popped up. It was reading a file on the 209,000th record, it was mainframe *APL*, but I was able to get the program to continue. I didn't have to rerun 209,000 records to redo it, and that's the big beauty, I think, of an interpretive language. So I've done both.

MR. TOBLEMAN: It's beauty is also a drawback if you're going to be using a repetitive, such as if you're doing some valuation every day, every week and it is very large. The beauty of it can also be a problem. You're exactly right, that's one of the trade-offs.

MS. TOOHILL: Yes. But I do have a question on *Magic*. It sounds to me like this is a great thing for the sort of stuff that *COBOL* might do, but is it a number-crunching language?

MR. KOFFLER: That's a good question. For example, an insurance package is actually in some of the big insurance companies around the country and is written in *Magic* and does lots of number crunching. But let's say, for example, you already have certain procedures written in *C*. I don't know *APL*, so I'm sure if it compiles—

FROM THE FLOOR: It does not.

MR. KOFFLER: Oh, it doesn't compile. But I could actually run any language that compiles to an executable from *Magic*. So I could use *Magic* as a front end and actually run some of these compiled executables that you already have in your

system. But can it do number crunching? Yes, it definitely can do number crunching.

MR. GILES: But it has all the mathematical functions; I mean even trigonometric functions I think, doesn't it?

MR. KOFFLER: Yes.

FROM THE FLOOR: My application is to produce for financial statement purposes some reserve numbers. The input into it is a set of assumptions for interest rates, termination rates, lapses and that kind of thing, and also a set of in-force data, the 109,000 people in force. We had a system set up in which *APL* took the assumptions and created a file of reserve factors. A *FORTRAN* program took these reserve factors and applied them against the in force and produced a file that had all the information in it. And then we took a reporting language, SASS I think it was, that took the information that *FORTRAN* produced and created a nice document that would go into the financial statement. This seemed like the best solution. Does it seem like the best solution to you, and what language should it have been in?

MR. TOBLEMAN: I'll speak quickly. I think that's a very intelligent use of the various tools. Your use of *APL* in that example is correct, I think, and I would stop it there. It's very good for that. Then let something else that's faster at applying factors to in-force units do those calculations and then finish up with a report writer that especially is nice and neat for designing reports. I think that's beautiful.

MR. CAVAZOS: Well, if you're doing that and looking at that, what would you do? Would you want, on an ongoing basis, someone to have three groups of people to learn different software packages? Would you want one language that could actually do everything that you need, from taking your database all the way to polishing off a report, or do you want people to essentially learn one language for this aspect and another language for another aspect? I know in my experience it seems like a lot of this has been evolutionary. We originally had a file database system and an evaluation system. Later we added on a report system. And then later we upgraded systems. Part of the system is now in *COBOL*, part of it is in *FORTRAN*; some that could be done in *COBOL* couldn't be done in *FORTRAN*. Where do you make a bridge of using one language, and how do you maintain the knowledge base for all the different languages? It seems like a problem to me.

MR. TOBLEMAN: The actuaries are primarily the ones doing the assumption setting and the factor derivations. And my personal belief is I don't think actuaries should follow the project necessarily downstream. Once you have reports and you have a program that will apply factors to in-force units, why does an actuary need to follow that downstream? So I think it's natural for other data processing people or other people within the company to have those other skills. I also think you need to understand that these other tools are just good tools in general. It's good to have a *Magic*-type program or something else, and that's not going to be the only application. So it's not exactly like we're going to be wasting resources because we have several different platforms that are being used within a company; that's my feeling.

MR. GILES: In the life insurance application of *Magic*, you must compute the policy's TEFRA premium, its seven-pay premium. They do that calculation downstream to a spreadsheet. It probably could be done in *Magic*, but one of the owners of the company, an actuary, says that most actuaries do it in a spreadsheet. So there's an example of mixing and matching, but I sympathize with Mark's point of view.

MR. KERRY A. KRANTZ: Frank's comment about what he does is similar to what I do. When we do a projection of, say, a five-year plan of where our business is going for statutory or GAAP reserves, if we start with our GAAP reserves as an example, our in force is over 120,000 records. So I'll run a *FORTRAN* program on the mainframe, which will project it for five years record by record producing summarized results. I'll then download that to a PC and put it into a LOTUS 1-2-3 worksheet. When we do our valuations, we have quirks. It seems like the factor generation is the easy part. The computer does all the work, we don't have to worry about it, but because it's easier in a spreadsheet to do manipulations, since each cell is essentially a program, you can do many things that way. I want to point out that you could use several different languages. You could use *BASIC*, *APL*, *LOTUS*, if you want to call that a language, or FoxPro or dBase 3 or whatever, and you can link them together because they have hooks now in the languages, user exits. So, for example, if you have a .BAT file, you can start where it will say, start LOTUS 1-2-3, which will have an auto 1-2-3 file, which will then load a spreadsheet, which can then have an exit, which can then run a *BASIC* program, which will create some of its data and bring it back into *LOTUS*. I find that makes things easy. I try to write a lot of macros; I don't have to worry about editing what's in the spreadsheet. It will automatically bring data that I've downloaded from the mainframe. For example, if I download a file that's called August 1993, then the spreadsheet puts in a section called August 1993.

MR. SLOAN: As a consultant in product development, I have to communicate with my clients easily, and we use *FORTRAN* for most of our product development work. But then I can take the results of my *FORTRAN* and import it into *EXCEL*, which I can then fax to the client. It is a very useful process. Now I'd like to call your attention to another language that I use at home; it's called *GAUSS* (Aptech Systems, Inc., Kent, Washington). You might describe it best as a left-handed version of *APL*. It works very much like *APL* starting with the assumption that everything is a matrix. And it's very, very fast, but it doesn't go right to left. It's a PC-oriented language that got a whole lot more out of the old 86s than anything else did. It's marketed mostly to statisticians, but it has hooks for *C* and *FORTRAN* and whatever.

MR. GILES: Let's get back to SASS, because so many people raised their hands on that. What's your view of SASS?

MR. KOFFLER: I used SASS when I was in college doing regression analysis and I haven't used it since.

MR. GILES: But is it a fourth generation, where does it fit in this hierarchy, and what is this hierarchy?

MR. KOFFLER: SASS has changed a lot since I used it. When I used it, it was mostly used for analysis purposes, and it at that time was less of a development tool.

From what I've seen and from what I've heard, it's become a lot more of a development tool, but I haven't used it in probably the last 8–10 years.

MR. GILES: Well, let me ask you another question. You say *Magic* is beyond fourth generation; can you take us through the generations?

MR. KOFFLER: OK. Let's look a little bit. Actually, at the lowest level is Binary, zeros and ones. Then you go to the next step, such as a 2 GL, which would be assembler, which actually interprets the Binary. And then you go to the third GL. I don't think anybody, at least in this group, deals lower than that. You have for instance *COBOL*. In *COBOL*, you're writing lines of code but you have to write quite a bit of information to actually get the result. So they decided, why do we have to write all this? Let's make this easier so they came out with 4GL, and usually 4GL deals with sequel or *SQL* with relational databases. It makes it easier to get to the information, and you're writing less information to get to it. And in a way I would say *Magic* is a post-4GL, because instead of writing lines of code and editor, you're describing what you want to do into tables and it makes it even easier. So it's just an evolution of the systems.

MR. GILES: What are the fourth-generation languages?

MR. KOFFLER: You have *Focus*, you have *Progress*, you have *Power Soft* or *Power Builder*, you have *GUPTA*; there are 10,000 different languages.

MR. GILES: And that is the problem we're facing; how do we sort through all these?

MR. KOFFLER: Actually, that's a good question; how do you decide what is the right language? You have to look at what the application is. What are you trying to have occur? What do your end users want? Decide which will be the easiest way. The best way I've seen to decide which tool is the one is to get a piece of a project that you already have and do what's called a rapid application development project. So say here's the project I've already done. I know how long it took me to do it in COBOL, for example. Say I want to do this piece, I know how long it will take, I'll write out specifications, and I'll give these specifications to maybe three or four different companies and ask them to come in and do this particular project on your system, on your database, using your data, so you see if it works. And that's the fairest and the best way I've seen to look at different tools and decide what the best tool is. Because it's your system, a piece of your system, you know how long it took, you're seeing it done with your data and on your system.

MR. GILES: I asked the few people I know who use *Magic* how they started using it. And their answer is, they try every new program language that's available. I can't do that, I can't drop *APL* and try everything. If everybody were like me, there would never be any progress, nobody would ever try something new. But that is the problem. When do you decide how to take that next leap? We're all struggling with that. Does anybody sympathize with that?

MR. ROBERT E. GOVE: I've gone through this for the last five years on what to do. And I learned *FORTRAN*, we didn't have *APL*. And when I went to PCs, I migrated to *BASIC* for the same reason Dennis likes *APL*; you can stop it in the middle, change

your data, and progress on. It's cheap but it worked, and I still have some programs I wrote ten years ago that work. I'd like to rewrite my valuation programs and I have been. I started doing some things for clients that they wanted done cheaply, and I tried to find a way that they could do all the input. And I went to Clarion because I could do the screens; I can paint the screens on the PC and it generates all the code. I can draw the report on the screen and it generates all the code. Clarion's come a long way. It just came out with its Version 3 last year and it has a lot of problems. I'm part of the beta testing on the WINDOWS version, it's a database management. If you people are in *APL* or *FORTRAN*, I'd stay there. He talks about an end user; the hardest thing I had to figure out was that I am the end user. I am the user, I'm the end user, I'm the developer. When he's talking about end user, he's talking about the person out there who's doing the policy work, inputting the issue date, or the address. That's not you, and I had the hardest time figuring that out. I've stuck with Clarion because it does all the screens for me. You mentioned *C*; I'd leave it alone. I've never had anybody who programs in it tell me anything good.

MR. GILES: We need more speakers like that; we want this to be confrontational.

FORM THE FLOOR: I have a question, I guess it's for Mark. I've used *APL* a little bit myself, and *APL* has been referred to as a write-only language. Once you write it, no one else in the world can understand what you wrote. And it seems like *Magic* kind of has some of those qualities, too. Everyone is free to set up his or her environment exactly the way he or she likes it. So when I come into someone else's environment and hit the F5 key, it doesn't do the same thing as when I get into my environment and hit the F5 key. If you were to go in and debug a program or try to modify a program, which would be the best language that you would like to look at?

MR. KOFFLER: Can I first just differentiate that a little bit? Remember we talked about the different tables. The first table lets you set standards so you can set standards for all my developers when they're developing the application. Here are the stands that they're going to do, this is the keyboard mapping you're going to use, these are the colors they can do. I can tell all my programmers to use this color when they're developing a data entry screen and they say, well, OK. Now the president of the company says to change this data entry screen. I can go to one place, one table, make a change, and it's changed throughout the system. That's allowing *Magic* to use its object-oriented features. Also, talking about documentation, *Magic* has full documentation. For everything you're doing in the system, you can get complete documentation, and also you can actually only document things that you want to document. So it lets you easily set standards. If you press this key and this is your standard and it's set on your application, every single place you make this change it will always be the same. Now if I went into a different installation, it could be totally different.

FROM THE FLOOR: So what you're saying is that the person operating it is setting the standard; it's some central location.

MR. KOFFLER: Well, actually it's a choice. It could be centrally set or you could have it so the user could set it. If the user sets it, then it's going to be different between two different machines; that is correct, right.

FROM THE FLOOR: One user sets the function key one way. I'm used to my way, then I come over here, it doesn't work.

MR. KOFFLER: So if your company wants to set a standard, you can easily say this is a standard and every single machine, terminal, if you press this key, will always do this. When you're testing your applications, you don't have to test whether these keyboards are going to do certain things, you know they'll always do that particular function.

FROM THE FLOOR: So I can set up my function keys a certain way, and someone can override them.

MR. KOFFLER: Yes, that is correct.

FROM THE FLOOR: If you come in one day, it doesn't work the same.

MR. KOFFLER: If they want to set the standard, they could, yes.

FROM THE FLOOR: I guess I'd still go back to Mark.

MR. CAVAZOS: Well, in my experience, I haven't done the actual programming. We've had a programming group, but the problem that I see is somewhat related to it. In some of the very old languages that we had, we had programs that were actually written in assembler language, and whoever the programmer was left and now it's a black box. It was just being used and we knew what came out of it but if something went wrong, we wouldn't know what to do with it. That's why to me it's not so much whether one program is truly superior, one language is truly superior, but that they're consistent for all segments of your program. That's what it seems to me would be important. Also, because programming people may not stay with a company forever, you will need other people to come in who already know the language. So I would think you'd want something that may be a little bit more universal and then within that, have the documentation for it.

MR. GILES: There is a trade-off between programmer creativity and documentation, and I guess these object-oriented languages drive the professional programmer crazy. It takes away his or her options.

MR. KRANTZ: Going back to my memories of *APL* days, I want to talk about Roberta Canfield, who was my boss at my prior company. She was the best *APL* programmer I've ever encountered. She would write programs that were table driven, which is what I would call a fourth-generation-plus language. Instead of having a long coded program, in which you have to look line by line to see what's going on, she would create tables where you would go into that table to find out what you wanted to do, and so documentation was automatic with her. She would just print out the tables. If you were calling the table that came up with lapse rates, you would just say "load table A." There would be your lapse rates, and the formula for calculating persistency would be right there in the table. So documentation depends on programming abilities. Sometimes in *APL* you'll have a long code and nobody will understand it because it was written A gets 3, B gets 5, C gets A+B. Well, that's easy, but when you go 400 lines and you've got all the Greek letters in *APL*, I won't remember

all these combinations of the characters, so the language is important. *COBOL* is a good language because it automatically forces you to document; *FORTRAN* is less so. *BASIC*, when I first encountered it, was a single-letter version of *FORTRAN*. I think today it's a lot improved. The only experience I've had with *C* is we had a non-actuarial programmer at my last company who programmed in *C*, and he explained it to us one day. It seemed like it was closer to *COBOL* than other languages and that you had to declare more things.

FROM THE FLOOR: My question is for Dennis. I've always had an interest in *APL* because *APL* seems like it's the actuarial language. The languages we use are *Easy Tree Plus*, *SQL*, and *LOTUS*. When we got *Easy Tree Plus*, it made things in my department just so much easier because I do things, such as calculate a raise. I guess what I want to know is, what are some of the health-insurance-related things that can be done in *APL* that can't be done in these languages, or that are much easier in *APL*? I'm still interested in learning about *APL* and wonder if our company should get it.

MR. TOBLEMAN: Well, I'll try to say quickly, first of all, if your company's needs are being met with the tools you have, why change? Now what you're asking is, are there things that you're not aware of that maybe could be served better? A guy I used to work for has written profit projection systems in Symphony. Now I've written one in *APL*. I can assure you, mine allows much more flexibility and input, it's also more complicated, but I've taken the time to put in more tables, a more on-line documentation that automatically comes up every time someone calls in a certain segment of the program. So you don't have to be looking for help keys and the like. Again, *APL*'s strengths are present-value calculations and projections, because you can do calculations of premiums and things like that in your spreadsheet environment, and I'm not familiar with *Easy Tree*, for instance.

MR. GILES: One of the first applications of *APL* that I used was disability income premiums. *APL* seemed to work well where you had termination rates and incidence rates and many tables. The Society is promoting Actuaries Online. Have any of you experienced that? Anybody want to speak to that, help the Society out?

MR. GOVE: If anybody in here is on CompuServe, it's just like anything else you get into, as in the forum. If you catch anybody on line, you can call him or her up and talk. It's easy communication; you can leave messages to many people. If everybody will get on, you can pull up and see if anybody needs help. You were talking about support; I use Clarion and *Magic*'s the same way, I'm sure. You go in, you have a question, you just leave it. You come back two days later, you have about 20 answers. You start a thread, and it goes off on different tangents, but with Actuaries Online, if you need something, you have specific questions, you can ask a question. I've already gotten some good help on it. I recommend it.

FROM THE FLOOR: I started my actuarial career by using *APL* and it's great. I started using it for pricing and calculating reserves, but *APL* is not that great for experience studies. Then I got SASS on the mainframe and SASS is so good for doing experience studies. I guess most people know, it's just using a summary to get whatever you want. We have SASS on PC. I find SASS on PC is quite slow. Then I moved to the current company. I don't have SASS; I have to go back to

FORTRAN. When I do experience studies, I do a state-required report. You always need a summary by policy form, by whatever criteria to get the premium, get a trend. *FORTRAN* seems very quick, but it takes too much time to program it, and *APL* is not that good. I tried to convince my boss to purchase SASS, but the problem is it's too expensive. A week ago, I had a chance to be exposed to Microsoft Access. You have to copy the data to memory before you can do anything, so I think it will have some memory problems. SASS on PC has memory problems. Is anyone doing experience studies, doing those kinds of state reports? What other choice is there except SASS?

MR. RANDEL S. SWANSON: For right now, we have a language called *Focus*, which is a database that we use to get in-force information off the mainframe. Then I take the information as it's summarized from *Focus* and download it to the PC. I put it into a spreadsheet and I do some stuff with it. And then I take the numbers and I shove it into *APO* and I do some more with it. So it's going back and forth, but *Focus* is the language that I use; it's another database manipulation. It just gets me a summarization of the in-force data and I shove it into *APO* or *LOTUS*.

MR. HOWLAND: *Focus* is also in use at Blue Cross in New Hampshire mostly by the systems folks. There's a little bit of a tug of war between the users who like SASS and the systems folks who like *Focus*, mostly because SASS wasn't their idea so they don't like it. But it is similar to SASS. I can't speak to the support that its organization provides, but I can also tell you that it's more expensive than SASS.

FROM THE FLOOR: I've done a few experience studies and use SASS on the PC, and I found that for the blocks of business that I've been looking at, time and speed is not a problem, it gets done quickly. I'd like to echo the expense problem there. SASS is not a software package that you buy for your PC; you're leasing it so you have a charge. If you're only one workstation in the office, it could be $500 a year instead of just buying a package for $300. We're looking at other databases; Manager, maybe Paradox, or something like that might be able to do the same kind of calculations and summaries and report generation.

MR. GILES: Let's get back to Clarion. We had a strong support for Clarion. Does anybody else use that?

FROM THE FLOOR: This is not Clarion, but this is a Paradox SASS situation. We use SASS heavily and we had one new programmer come in and use Paradox. I wouldn't do it. It isn't as efficient, it was much more problematic. I do have a question though. We seem to take our health care data beyond just experience studies. We're often using large health data files and other mixed sources along with perhaps demographic files, doing a lot of mixing and matching. I'm told that SASS doesn't do that well and that we need to move onto another product, a relational database manager or something. I just want to know if anybody else is doing that sort of thing. Is there a good set of tools for that?

MR. KENNETH K. LAU: I'm partly responsible for developing a large software product called Champ, which does health care monetary. We at one time started with Clipper and then we moved to FoxPro. Now we just changed it to *Power Builder*. Sometime early next year there will be *Power Builder*, which will be WINDOWS based, and

a relational database at the background using client server technology. This is talking about developing something for a user-oriented system but if you want relational database, Paradox is a good one for desktop, or FoxPro will be a good one for desktop if you do programming. For end-use oriented, I think Paradox is much better than FoxPro. You don't have to worry about the language, you just use the Query By Example (QBE) and you just pick the field that you want, bring up two or three databases, and put them together. It's very easy. For example, you take the demographic data on one table, and you take another piece and your experience data on this other table, bring them together, and it's no problem.

MR. KOFFLER: Can I say something too? *Magic* does allow you to get all this data, and that was one of the major features in the beginning. All these islands of data and *Magic* would help you bring them all together and use all the information so your end users can quickly have all this information and make use of it. And if you do look at tools like that, make sure you look two ways. There are more than two ways, but there are two main ways to get to this data. One is what's called ODBC. ODBC is an open way that allows you to easily access data, and then there's what's called optimized gateways, and optimized gateways is written specifically for that database. Now the differences in an optimized gateway will be much, much more efficient. It's written for a specific database rather than ODBC, which is more general. Of course, *Magic* is optimized gateways. So when you're looking at this, take a look because some tools will let you access data but they use ODBC. If you're using large amounts of data, you'll have speed and transmission issues, because a lot of it will happen on the client side instead of on the server side. When we start dealing with the client-server environment and relational databases, those issues start becoming very important.

FROM THE FLOOR: I use *APL* because the tasks I use are basically a lot of quick and dirty programming, and I appreciate that there are documentation problems with *APL*. But because of what we do, it fits our bill. However, we've talked about the weaknesses of *APL* and the weaknesses of some other languages; what's *Magic*'s weakness?

MR. KOFFLER: What is *Magic*'s weakness? Say I was going to write a device driver, for example. One of our clients has have gas stations all across the country. It deals with the pumps, sending information back and forth to the pumps. If I was going to work at that kind of low level, I would write that in *C*, and I would use *Magic* as a front end to call it. Besides that low level type of thing, I see all types of applications written in *Magic* from artificial intelligence, many business applications integrating with financial systems and insurance packages. Our other major weakness is that we do have a good graphic user interface (GUI) product, but it doesn't have all the features yet that a product like *Power Builder* does. We have a product coming out at the end of the year that will have more of those features.

MR. GILES: A GUI feature being?

MR. KOFFLER: A GUI feature being where you're using a product under WINDOWS. WINDOWS allows you to have many, many features, and *Magic* allows you to run as a WINDOWS application, but it doesn't allow you as many features under WINDOWS as a product like a *Power Builder.*

MR. LITTLEFIELD: Someone mentioned linking a lot of data together, and that sounds like something that's not on your PCs. Is that correct?

FROM THE FLOOR: Mainframe.

MR. LITTLEFIELD: We're in a situation where we're not a big company as far as number of actuaries. We do deal a lot with the information on our minicomputer and there's a distinction. You have tools that you'll be using on your mainframe. You have large data sets, you have millions of records, you don't want to fiddle with that on your PC. You have to have a tool on your mainframe to deal with that sort of thing. You have to extract it and/or summarize it. There are tools out there for doing that and for linking raised data sets so that you can link your enrollment information to claims information, to anything else that needs to be linked, and you do all that beforehand, create a suitable file that is more manageable. Those 100,000, 40,000, 10,000, 5,000 records that you get down on your PC and you can play with it at a database, you can put it in *APL*, you can throw it in *Magic*, whatever. But keep in mind that if you have that data up on your mainframe, you must have something to deal with it. You need a large computer, and that's something we haven't touched much, but that's a very important consideration, what you're doing out there. You save a lot of grief when you put it down in your PC. Give some thought to what you do up there as far as putting all that data together. They have some nice tools. We're on an HP platform, and you have PRESENCE and some other things you can use to link data sets, even if you don't have relational databases. You don't have to get a relational database if you want to link things. There are tools out there that do that on your minicomputers or your mainframes. You link it up there and then you have a nice file you can use on your PC.

MR. GILES: Well, we are shooting at a moving target because as the PCs get more and more powerful and can handle more and more records, the equation keeps shifting.

FROM THE FLOOR: Have you ever heard of *Pac Base*?

MR. KOFFLER: I haven't heard that, no.

FROM THE FLOOR: I knew a little bit about it, like maybe five, six years ago, but you set up tables similar to *Magic*. I think it originated in France.

FROM THE FLOOR: We moved toward office of the future. We have remote users trying to link via modem to the local area network (LAN), and then we have people in house on mainframe. We're trying to move data all over the place, and one glitch we've run into most recently, as we have now some permanent remote users, is trying to get into the LAN in a WINDOWS environment. Is there a way to get into the LAN when you're in WINDOWS?

MR. KOFFLER: Sure. Many products out there will do that. Many communication products will let you easily connect from a remote location. Carbon Copy is one of them.

FROM THE FLOOR: We do that but not with WINDOWS. Apparently, WINDOWS takes ten minutes to fill out your screen when you're using Carbon Copy. We had to have $10,000 for the right telephone line.

MR. KOFFLER: Yes, a T-1 line.

FROM THE FLOOR: We wanted to avoid the $10,000 telephone line.

FROM THE FLOOR: It's my understanding that the DOS system—and we're not doing commercials for any system, but since we at our company use it, we need to know these things—is going in the future to run off of the WINDOWS environment. And because we have a LAN network, I know that we have to be able to access the DOS system through our network. We had problems with WINDOWS when I upgraded from four meg to eight meg on my PC, and so we installed 3.1. I know that the networks in WINDOWS do talk to each other, because when they installed WINDOWS, there was a question of whether a LAN was present. We're on a Novell network and it recognized the network.

MR. KOFFLER: If anyone wants information on *Magic*, the number is 1-800-345-6244.