RECORD OF SOCIETY OF ACTUARIES
1993 VOL. 19 NO. 2

TRENDS IN TECHNOLOGY

Moderator: STEPHEN J. STROMMEN
Panelists: LARRY A. CURRAN
MARK T. MCANDREWS
MATT TUCKER
Recorder: STEPHEN J. STROMMEN

New technological trends are emerging every day. But after an extended period of confusion due to the multiplicity of competing directions, some current trends have been brought into focus. To that extent, development using these technologies can be undertaken with confidence. The three panelists relate recent personal experiences when emerging trends in hardware, software, and methodologies have been clarified.

MR. STEPHEN J. STROMMEN: We'll be talking about ways of using today's technology in order to further actuarial applications and actuarial projects. We have three panelists who each have been very closely involved with that kind of work. The first panelist is Larry Curran who has been with Northwestern Mutual Life Insurance Company for over 30 years. He is responsible for the actuarial systems division (ASD) and is an officer who is responsible for the group that supports actuarial applications. Larry will be talking about the use of systems from the viewpoint of someone in a large life insurance company.

Matt Tucker will be speaking next. He has been at Tillinghast for over 20 years. Mr. Tucker is responsible for the actuarial aspects of Tillinghast's actuarial software. He will be covering the use of technology from the viewpoint of an actuary in a large consulting firm. He has some interesting topics, one of which is a way of allowing actuarial consultants essentially to develop software without programming. He also has some views and experience using other than Intel microprocessors, which I think will be interesting.

Mark McAndrews is with Chalke, Incorporated. He has been there for two years and works closely with the liability side of its projection software. Mark will be talking about object-oriented operating systems, one of today's hottest buzzwords.

MR. LARRY A. CURRAN: I'm going to cover several topics, most of which are somewhat interconnected. My topics relate to how we're making technology work for us in the actuarial department at Northwestern Mutual, and I probably will not be getting into exactly what the technologies are to any great detail.

The ASD is part of the actuarial department and consists of approximately 20 people. This division has been growing slowly over the years. These people are systems professionals. They have math backgrounds. Most of them are math majors in college, generally with courses up through at least calculus. Until recently that was our primary requisite -- someone with a strong background in math. In the last few years we've become more concerned that these people have systems background as well because, with the rapid change in technology, we found the startup time is too slow if they don't have this type of background.

The training for the company is done by the information services (IS) department.

The ASD people really are pretty well set with their counterparts in IS. The advantages of having a group of professionals within the actuarial department are really great. The primary advantage, of course, is that you have direct control over your IS resources, and of course, these people tend to respond very well to the demands of the actuarial department because this department does their evaluations and pays their salaries. They know what to do. However, they also have very close ties with the IS department. The IS people are the first people they meet in the company. They go through training with them; they retain close contacts with them; and they have a very good working relationship with them.

Communication with actuaries is very good; they work right alongside of the ASD. Because of their math background, the ASD understands the nature of the actuaries task, and it gives them the opportunity to translate what the actuaries say to the IS people.

The ASD provides support to the actuarial department with regard to valuation, actuarial studies, reinsurance administration, our 401(k) savings plans, and our retirement plans. These people support the areas of client-computing within the actuarial department. They provide technical support for automatic premium loan (APL). They also provide services to the IS department -- a rates and values routine, which I'll be talking about later in more detail. They also provide on-line values in the form of dividends to policyowner services, or any other insurance department, and they do a great deal of systems checking for the IS department, primarily in the area of sales illustrations.

Rate routines are something that, for several years, I thought were unique to Northwestern Mutual. I've noticed that several of the companies that now are developing administration systems use a form of rate routines rather than factor files. We started out with the rate routines back in the 1950s when we first got into computing because we didn't think of any other way to do it. We had a rather simplified product line at that time, and we were able to fit the rate routines quite easily into our system. Over the years we've found that it's provided us tremendous flexibility. It allows the actuary to develop any kind of product. There are no restrictions. We're not restricted by factor files at all. Particularly in dividends, when it came time to vary dividends by interest rate, it was easy; if there was no change to a factor file. All we had to do was make a few minor programming changes, and we had the new rate in there. Similarly, when we went through some major update programs that affected large blocks of business, it again was very simple to accomplish this in a relatively short period of time.

Things have been expanding in the last few years, and we've taken on a little more, too. Now, we calculate also the actual dividends and multiple-use routines that are used throughout the company, and we'll be expanding more into other types of value calculations, like cash values and so on.

We started using PCs in the mid-1980s, and at first, we weren't sure if they were going to be of any use to us. We found they were good for word processing and a few spreadsheets, but we really didn't have much use for them. In the mid-1980s we were going through the development of a new series and found that our pricing programs were using extensive central processing unit (CPU) time on the mainframes.

We were getting a lot of pressure from the IS people, who didn't understand what we were doing and why we didn't take these programs and put them in production by running them once a week. They thought we were testing, and of course, in a sense, we were, but not in the way they thought. We had a lot of mainframe capability at the time. IS said to not ever do this again.

They let us get PCs, and we got STSC APL at that time. First, we downloaded the APL programs from the mainframe. That's when we discovered why the run times were so long: we were recalculating everything every time for no reason at all. We did a little bit of cleaning up on it at the same time.

In the last year we've decided to make better use of what is coming out of our pricing programs in terms of the overall support of the actuarial department. One thing we decided is to use the outputs from the pricing programs as an automated way to check what we are doing in the rate routines. At this point in the company, nothing is calculated by hand – in fact, it couldn't easily be calculated by hand. Everything is done in floating point, and our checking procedures are very effective at this point.

What we're looking for in the future – I'll have to admit we're a ways from it yet – would be for the pricing programs to provide some type of feed into the administrative systems. By this I mean getting at possibly product rule tables being set up and the pricing programs providing the feeds to them. How far we can go is still an open question. But we have some people who are pretty serious about looking into it.

Regarding PC rate routines with sales illustrations, we had factor files again and lost our flexibility. We decided in the late 1980s that we could fit the rate routines on a PC, and it turned out we were correct. We replaced 79 diskettes with seven diskettes, which was one of our selling points. The actuarial department's selling point (kind of a side benefit) was us providing flexibility that IS was losing. This brought a lot of PC expertise to the ASD which became one of the leaders in the company in terms of understanding utilization of PCs.

We didn't think much of the idea of local area network (LAN) when the company put it in as a test site. We had a lot of problems with it. But, once it functioned, we realized that it was impossible to live without one. In the beginning, all we did was share software. Now, we're getting into sharing information as well, and it provides a gateway to the mainframe, which is far more efficient than anything we've had before. It gives us a very effective way of downloading information from the mainframe in a relatively quick manner.

For us, Windows is the new hot word. It's driven a lot by client demand. Anyone who has seen Windows does not want to use anything else. We're planning on any software that's to be developed within the actuarial department for PCs will be done for Windows. We find tool selection for that development to be very important. The company has selected Visual Basic as the overall tool. We think that we probably will be using Object Windows. We will be moving more towards object-oriented technology, and we think that might be a good selection. We're not really all that set on anything at this time except Windows, which is the standard for the actuarial

applications at this point. We just cannot see developing things for DOS any more; we don't think that we'll be using it that much longer.

The languages that we use in the actuarial department include, of course, APL. We have some actuaries using Basic. But, APL is the standard language and seems to serve our purposes quite well. The systems division has traditionally used PL/1, which is the official programming language of Northwestern Mutual. However, when we went into the PC environment, we used C. The people using C thought it was the greatest language that they had ever seen. But many people thought it was rather a strange language, and I'm one of those who has had some difficulty adjusting to it. However, it turned out to be a very good choice because this is a language that is going to allow us now to cross platforms. We can develop applications on the PC, bring them up to the mainframe, and run them there. We have a SSAS C compiler on the mainframe. Currently, we're testing a C++ for the mainframe; C++ is the language that we will be using for our object-oriented technology.

A month ago, object-oriented technology was declared the standard for the ASD. I expected to run into considerable objections from some long-term people, but much to my surprise, the reaction from everybody within the division was extremely positive. People huddled around trying to figure out how they could get in on it, what they could do, and it was a very good sign. I think it's well-suited for actuarial applications because we need to share what we do in terms of our use of codes that could be reusable. I don't really want to get into object-oriented technology, which is a separate topic, but one of the things that it's done for us has been to have breathed a new life into the systems division. It has also created some effective partnerships that we didn't have before. We've established some strong partnerships with IS in terms of looking at what each other is doing, sharing our code, and so on. We've just formed a partnership with tax and financial planning, of which Steve Strommen is a part, that I think is quite strengthened because of getting into object-oriented technology.

Preparations for object-oriented technology are extensive, which I think is the drawback. We don't have a good way yet of training people in object-oriented technology at our company. We learn through experience. In terms of the language, we simply send people out at $1,300 a crack to train them on C++. We've had a couple people who have been working extensively in object-oriented technology for about a year, and I asked them if they're experts. Their eyes just kind of glaze over, and they say, "Well, we know a little." That worries me, but I don't know if anyone admits to being an expert. I think in fact they are telling the truth, they are still not experts.

We've found it to be very important to have an IS partnership to get the actuarial work done. I'd like to say that I'm the one who really created this, but unfortunately, that's not true. I think the IS partnership was developed primarily through the systems professionals within the actuarial department. By working with their counterparts, they've helped get the message across that what the actuaries do is an important part of the company's operation. We've reached the point where we do much sharing of personnel resources. One of my top people spends 50% of his time working within the IS department, helping with data modeling, strategic planning,

technology planning, and in turn, they provide us with free interns for the summer or any other type of help that we might want in terms of professional assistance.

We also have been invited in on some projects. We're considered now to be the multiple-use experts, and we are heavily involved in developing multiple-use routines for the IS department. Our object-oriented technology, of course, is being coordinated very closely with them. They're looking to us to provide a little leadership on that.

Of the topics covered, I think in terms of how technology can work best for the actuarial area. The most important is to form a strong IS partnership, which is difficult to do. In our case having an ASD was very helpful in getting to that point. I think the rate routines have provided us an edge that is hard to match, and we've been very pleased with that. But, of all the things I covered, I think that the IS partnership is perhaps one of the most important things.

MR. MATT TUCKER: I'd like to start with a quote from George Bernard Shaw, not a name that you probably associate with technology: "You see things and you say why, but I dream things that never were and I say why not." I think that should be the creed that people in technology should take. Unfortunately often that's not exactly their view. Often they're like all the rest of us in preserving the status quo, and certainly in technology, status quo only lasts for a fleeting moment.

I began in software more than 30 years ago before, probably, many of you were born. I wrote my first program in 1959 at the University of Texas in my last year in school on a machine you've probably never heard of, an IBM 650. It was a rather exotic computer. It used a language called SOAP. I'm sure no one has ever heard of it – Symbolic Optimized Assembler Programming II – actually version two. It was a really good one. This machine had 2,000 characters of 10-word memory locations, and it was not RAM as you would think of it today. In fact, the memory of this computer was stored on a drum that rotated inside the computer. The optimization part of it actually placed your instructions on the drum so that, as it rotated, your next instruction would be close to where it could be read into memory when it was needed. It was pretty interesting, I suppose, at the time.

I think it's safe to say that things have changed a little bit since that time. The three topics that I'll be discussing briefly are: (1) the evolution of computers in a consulting firm as I have experienced; (2) a description of a computer-based tool kit for developing actuarial software and our initial experience in using such a tool kit; and (3) hands-on experience at porting software to different hardware and operating systems environments.

In the early 1960s, access to computers for actuaries was somewhat limited. In fact, most of the calculations were done by staffs of actuarial technicians who had desktop calculators. In this case, desktop meant that it covered the entire desk; they were quite loud, too. These people would pound on these calculators all day to do all the calculations, asset-shares checking, cash-value calculations. They used real spreadsheets; you would go and look at the spreadsheet, and it would have little red dots indicating somebody had calculated the number a second time to make sure it was right.

In the mid-1960s, we acquired our own mainframe computer. We actually had rented time at night using a mainframe computer immediately prior to that. We used Cobol and Fortran. There weren't a lot of languages available at that time. But, in the late 1960s, we began using time-sharing services, which finally gave actuaries a direct access to computers.

In the mid-1970s, our firm acquired it's own time-sharing computer and used leased telephone lines to connect offices together. All of the actuaries in the firm had access to a computer, and some actually used it. We developed software and used FORTRAN as a programming language and APL, which were the two primary languages. Software was still typically actuarial software, cash-value calculations, and models. In the late 1970s, universal life began to emerge, and my firm had some hand in that. Our software expanded to allow for the new product with some of the complexities of that product. In that time, the hot topic of the day was structured programming -- top-down development. Actually, some of those ideas have survived as opposed to some other buzzword situations where not much has survived. We applied these concepts to our software development process.

Acceptance of the PC in the mid-1980s was a rather slow process in our firm, as I'm sure many of you have experienced. It was even somewhat volatile from time to time. Many people were quite comfortable with the apparent sense of control that a central computer gave them. I don't mean that in a negative sense because most people felt like the central computer would assure a high quality of software and consistent presentation to consultants and clients. In fact, the PCs of the mid-1980s were seen as Pandora's boxes, so to speak, resulting in a loss of control of this quality of the software. In spite of those feelings, we began experimenting with PCs and actually, of course, applied our structured-programming techniques to developing software on the PCs. Those of you who developed software on PCs back then will know that in many cases it seemed like a step backward because the tools on the PC were not very stable and not very advanced, in comparison to the programming language capabilities that were available on mainframes or time-sharing systems.

I'm going to discuss now briefly the actuarial development tool kit. In the mid-1980s, as we began to redevelop our software, we took that as an opportunity to step back for a moment and look at what we had done. After cringing awhile, we said, well, maybe we'll do a few things different this time. So, we decided at that time that we would use a little different approach for our models. Previously we had used what I would call a traditional actuarial approach in that our software in models would process each cell through the timespan of the model and add those results to accumulations. When you finished processing all the cells, you had the answers to your models. We decided that perhaps we should rethink that, and in fact, we decided to now construct our models to better mimic reality, and that would be to let time pass through each cell concurrently. We realized that would strain the limits of memory of computers. But, we certainly hoped that PCs would stay ahead of us, and certainly PCs have changed since the mid-1980s.

We also saw that we would need to streamline the development process so that we could react more quickly to situations where changes have occurred, and did we foresee anything like the changes that we have now? Of course not. But, we did decide that we needed to develop this software development platform to give an

actuary direct access to actually developing a system by interactively designing screens, making menus, writing formulas, and creating reports all within one environment.

Software is simply the process of transforming some data from one form into another form, and if it's all that simple, why is it so difficult to develop software? Well, in actuarial work this most often means transforming the data, measuring exposure and probabilities with some financial impact through formulas into the results that we're interested in examining or looking at. Actuarial software takes input exposures and probabilities, applies time value of money formulas, and produces the financial effect of those calculations. Now, obviously that's a slight oversimplification. But, that's essentially what we do most of the time.

At the dawning of the computer age, just a few decades ago, actuaries were really anxious to get at the power of these computers. However, along with computers came programmers. They too often spoke and still speak a language that's foreign to us actuaries as our language is foreign to them. To get access to their capabilities of computers we obviously had to speak with programmers, and they had to speak with us. We had to explain our mathematical world, which at the time was foreign to many of the programmers. It's maybe not as foreign to software developers now or programmers. As computers evolved, however, we did get access through time-sharing, and then we got to learn programming languages, in addition to our actuarial studies. We still dealt with programmers to get information from administrative systems. But, Larry touched on a situation that certainly sounds very good in terms of a relationship between an actuarial department and the IS department. It's not always that way.

Other areas that programmers and actuaries actually agree on is testing. Testing traditionally was a collaborative effort between programmers and actuaries. All too often I would hear something that ended like this, "But I thought you were testing that." So, it was never clear as to who would test what. The programmers thought they were through if the program ran to termination or end of job. The actuaries had a little different requirement for the program actually operating. They both certainly agree at how much they love to do documentation, a favorite of everyone. You've probably heard stories about programs that were amazing. But, they had to be rewritten when the author passed on either to a promotion, another job, or literally.

Finally, software that's created must be maintained. Actuaries and programmers agree here, too. They're especially fond of maintaining software and even more of maintaining software that's created by someone else, their predecessor.

So, we began developing this actuarial software development tool kit to actually help us develop our own actuarial software. We used the very complex software model mentioned before, input/transform/output, simple software. The basis of this tool is an input engine, a calculation process, and an output engine. The input and output engines we decided would have to be data driven or database driven. This would allow the software development tool kit to create the data necessary for these input and output engines. It allows the software to actually have dynamic menus, dynamic data-entry screens and very flexible reporting options. The engines are written in C,

as is all of our software. We selected that because of its portability and the fact that it was a compiled language and would run as fast as we could make it run.

We also needed to be independent of computer platforms as much as possible without spending a lot of time and money doing that. The input database engine contains lists of the input variables that are created within the tool kit. It describes the characteristics of all of the data residing in the program and where the data reside in the program so that other parts of the program can access the data, where the data appear on screens or menus. When they do or do not appear on screens, we actually use masking technologies to make the screens dynamic from the user's point of view. We have several data items that are included in the system, the typical being numeric scalers multidimensional tables. These are all standard items that are supported by the input engine, and the actuary developing the software need only select the type of item he or she wants and then put in the text associated with that item and so forth.

Option lists are another type of data item we have. In our environment the option list allows the user to select the type of information he or she wants based upon an option list that in shortened English displays the information the user can select or the particular options he or she wants. For example, the answers to a question would be yes or no in most cases. Maybe is not something we'd put in there. The type of reserves for a deferred annuity would be "issue-year commissioners annuity reserve valuation method (CARVM)" or "change in fund CARVM," and that's the text that would actually appear on the screen and in the tool kit.

The screens and menus are all drawn with the screen-painting process built into this tool kit. If you add fields to the input or menus to the screens, that's immediately added to the database. It's immediately available for inclusion in formulas, as I'll describe in just a moment, or it's available for the output database.

The output driver, engine, is very similar to the input, just the flip side. Again, you draw the output report and select the items that you want on that output report, and it generates the necessary data for the output engine.

The transform generator, of course, is our formula development part of the system. So, we decided to make it look very similar, at least, to a text editor or word processor in the way it works. You can put your formulas in. You can put comments and documentation all in the same sets of formulas, and you can have multiple sets of formulas open on your screen at the same time. As you add variables to the calculation process, they are added to a calculated variable database and are immediately available to be used for the report process, too, or other places in your actuarial formulas. We also have a library of standard actuarial functions, as well as any functions that the actuary might create for use in his or her systems, and you can just select those functions and use them in your modules or your formulas.

The system can actually translate those formulas into C language modules. We then compile and link them into a program or a library, depending upon the particular desires at that point in time. Our system also provides for an interpretive language so that at run time users can make their final adjustments to the formulas, or they can actually look at modifications to the formulas so that, at the time that they run,

without having to go back and generate the system again, they can make modifications to the formulas. This interpretative language looks strangely similar to the same language that you write your formulas in. Everyone would be surprised at that.

We've been using this tool kit in its entirety for a little over a year now in our software development activities. Additional items included in the tool kit are the direct production of documentation, doing the creation of the menus and screens, and you can provide on-line help. You can document what you're actually putting on the screen. When the target system is generated, then that documentation is available through a help process that's included in the input engine. So, as you create your screens, you can write the documentation, and it's automatically available to the user when he or she runs the system. The tool kit prints the formulas in a format that's a little more readable in the way that you actually enter them. You're limited obviously in entering them to a screen. But, a printed page gives you a few more columns of data and so forth.

There are several cross-reference lists available from the system for both input and the calculated variables, so you can get a complete cross-reference of where data items are used in case you want to check out what's happening to a variable in one particular area in the formulas. We include also a regression-testing process, which allows a new version of the system to be compared with a previous version of the system by the computer. This highlights the differences between the two versions, and so, you can quickly look at what changes have been made in the system, both intended and unintended. The tool kit also provides version control, such that at any point in time we can actually generate a previous version of the system directly from the tool kit by telling it we want a previous version, and we specify which that is. The version control is also used to make sure that, as a new version of the system comes out, previous data from previous versions of the system are compatible with the current system.

So, in summary, we seem to be headed, at least in our firm, to allowing direct development of the systems by the actuaries. There's limited involvement by programming staff. In theory some day maybe the computers will talk directly with our minds. I'm not sure that's good. But, the actuaries then can spend their time and brainpower solving actuarial problems and not having to get too involved, at least, we hope, in the software development aspects. The tool kit provides for multiple platform support, and I'll talk about that in just a moment. The actuary should have complete control of the system in this environment. In addition, this took kit provides control and documentation of that. This tool kit also provides a consistent look to software, and as upgrades come along, it can be applied to preexisting systems, so that you get the capabilities of newer features that are available.

You probably haven't noticed, but there is a fair amount of hype in the area of hardware and software out there. So, we decided we needed to get a real handle on what was real and what wasn't, at least in terms of our own software and what kind of speed improvements we could really expect as opposed to the benchmarks that you see published by all the vendors of everything. While we certainly haven't exhausted all the options, I think we only succeeded in exhausting ourselves at this point -- and our funds.

We've looked at several different things. The first I'll talk about is what I've called the replacement processor option, which involves installing a board on the PC as a specialized processor, which in all likelihood has it's own memory on the board that you install in the PC. We actually tested one processor, the Intel RI860, a RISC processor. We actually tested two of them. We have two different versions of the chip. It, of course, needs its own C compiler as you'll see in all the other environments. Each environment needs its own compiler. But, it was written in ANSI compliancy, and we had little difficulty in using our software directly and porting it to it. We use as a benchmark a 486/50 DX2 with 128K of external memory cache, and we had a suite of tests of our software to actually test different environments of it. I'll just give you the overall results.

The I860 with 16 megabytes of RAM cost about $4,000. It may be cheaper now because the vendor has a newer chip out. It's been out actually for a while. That's the usual cycle. As soon as a new one comes out and gets available, the others go down in price. This particular chip or environment seemed to run our suite of programs about 25% faster. We had expected greater improvement, but that's all that we ended up with.

The second area we looked at was numerical processors that replaced the numerical processor chip on the 386s, which is a 387 chip or is built into the 486 machines directly in the 486 chip. We actually tested two versions of the chip. It's the Weitek Coprocessor. Here we were disappointed even more in the sense that we only got about a 5% or 10% improvement on our benchmark over the 486/50. The chips, however, cost somewhere around $500 to $700, and there are newer and faster versions that they tried to sell us. But, at this point we decided not to buy it.

Finally, we looked at Unix Work Stations. The pricing has come down on Unix, and they are pretty competitive. We have one in house, a Sun SparcStation. It's a 30 megahertz Sparc 10, I believe, connected to our Novell Network. After compiling it on the Unix C compiler, we found that we had a performance of slightly less than 2 1/2 times faster on the Unix system, and this system cost about $10,000 by the time you get the operating system and everything to actually run the computer. If you haven't dealt with work stations, it's a totally different world, and Unix is definitely a different world. You certainly can't go to your favorite computer store and find a Unix work station or anything to support Unix.

Again, sure enough, hype bound, the vendor promised us that its $75,000 box was 15 times faster than the one we were using. So, we began to dig around a little bit before we actually plumped down any money for this box, and we found out that machine was actually an 8-processor machine with 50 megahertz chips. Our original box that we have is a 30 megahertz machine and the floating point benchmark the vendor used all 8 processors to show a 15 times improvement. So, if we revised our software to use parallel processing and used all 8 processors, we could get 15 times improvement. But, if you ran it on only one processor, it would be only about twice as fast as our machine or five times faster than the DX2/50.

But, there are work stations coming out that are one processor machines in the 100 and 200 megahertz range already. The Alpha machines that you may have read about from DEC are in that ballpark. Their pricing is similar to the pricing of this

8-processor machine. We have a Hewett Packard machine on order. That's something you'll go through if you order work stations. It takes a long time to get one, and it's supposed to be, according to hype, a really fast-floating point machine and we'll find out within the next month or two. This particular machine is about a $15,000 box.

So far, what are the conclusions that we've come to? Basically, our conclusion is the best you can hope for is to get what you pay for.

MR. MARK T. MCANDREWS: In the early to mid-1980s there was somewhat of a concern raised by the computer press and some of the users in the computer user community. That concern was that one software vendor would be able to completely integrate the suite of applications and thus be able to dominate the software industry. The two leading candidates were Lotus with its Symphony product and Ashton-Tate with some sort of combination of its database and word-processing software. These concerns proved to be unfounded.

I think the failure of this to materialize was not due to the lack of integration, at least on Symphony's part, but that the applications in each piece of an integrated application were being compared to the best available stand-alone applications. The people weren't ready to make the tradeoffs to take a less powerful word processor to get the integration or to take a less powerful spreadsheet in order to have the integration. So, I think that was the failure of the market there. The motivation behind it, though, was that the increased productivity users would have, if they had truly seamlessly integrated software, would allow a vendor to dominate the market.

I think we're on the verge of having that integrated software, and I think the glue that holds it together will be object-oriented operating systems. The object-oriented operating systems that are available on the market today include Windows, OS/2, MacIntosh, and Next. Now, the MacIntosh and Next are not available on Intex Intel architecture at this point. Next is working on that. So my comments will deal mostly with Windows and OS/2. I have a bit of a benefit here in that I get to talk about and describe the dream, rather than what I've actually done.

I would like to describe briefly two of the major facilities of object-oriented operating systems, object linking and embedding (OLE) and dynamic link libraries or (DLLs). Those give you the primary functionality that I think will be important. We're reaching a critical mass in the marketplace. So, I think in the next year or two years you're going to start seeing a lot of applications that exploit these and give us a great deal of productivity. I'll also finish up with a practical example of how actuaries will actually use these tools to increase their productivity.

Before I can define OLE, I need to take a step back and give at least a high level definition of objects. One of the principals behind object-oriented programming is the principal of encapsulation. Simply stated, that says that we can't really divorce the data from the programs. It needs to be viewed as a comprehensive whole, thus a word-processing program and the document it's working on would be considered as an object or a spreadsheet program, and the data it contains would be an object. I think an example here might clear things up. The definition of OLE then, would be the ability of one object to be tied to or completely contain another object.

As a couple of examples, one problem that often occurs is in word processing when you use multiple fonts in a document, and then the document needs to be passed to other users for review or modifications. If those other users don't have all the fonts available, you end up with the spacing being messed up and page breaks where you don't expect them. When the document gets back to the person who initially created it, there's different fonts than the author had originally sent it out with. So, one of the uses of OLE is the ability to embed the font within the document. Thus, for the users of the word-processing program, once they have a document that had fonts embedded, even though they didn't have those fonts available in their machine previously, fonts would be available within that document. Another use would be currently using math columns and mathematical capabilities of most word-processing programs. It's a bit clumsy and limits you in the mathematical calculations you can perform. What you really want to do, in most cases, is present spreadsheet-type data in a word-processing document.

What OLE allows you to do is, when you get to the point where you would have to be creating those columns in the word-processing document, you can open the spreadsheet from within the application. You can start it up. You can enter data. You can load an existing spreadsheet and just display a portion of it. You can format columns, change their width, and apply the formatting characteristics of the spreadsheet. Then, exit the spreadsheet and continue with the word-processing document. This allows you to really use the best tool for the job without having to jump through all kinds of hoops to have that available to you.

One of the benefits of OLE is that this integration really occurs at the operating system level, rather than at the application level. Currently, in DOS applications many of the word processors allow you to link to a variety of the popular spreadsheets. The problem is that the spreadsheet that it's linking to has to be recognized by the word-processing program, since it's handled by the application. If there's a new spreadsheet that's developed or new versions of things that change file formats, you may not have those same links in place. If it's handled at the operating system level, you not only work with all OLE-compliant spreadsheets, but also you have any that may be developed in the future. You're also not limited just to spreadsheets, but you could also include graphics programs from Harvard Graphics or numerous other things. So, you could drop anything that is OLE compliant into a word-processing document, or you could drop word-processing documents into a spreadsheet.

The key benefit here is this really shifts the focus from one of the tools you have available and how you get your results out of that, to the result you have in mind. You can use the best tool that's available for the job. Another benefit is that the links are dynamic, and they're two way. In current DOS software that links a spreadsheet and word-processing software, it's often a one-way link. Or if it is a two-way link, when numbers are updated in the spreadsheet, they get updated there, but you're not going to see them flow through the word-processing document until the next time you load it. With OLE you could have two sections of a spreadsheet displayed in a word-processing document, one section containing assumptions and the second section containing results. If you changed an assumption, you would actually see the recalculations occur and the results flow through in your word-processing document without exiting and recalculating spreadsheets. That's a tremendous benefit.

I want to distinguish between linking and embedding of objects. It's rather a technical difference, and the practical effect is you have the benefits in either case. Linking allows you to create links to existing objects, like spreadsheets and graphics packages, without actually incorporating them in the document you're creating. The benefit here is that you reduce duplication on a disk. The reason you may choose to embed something is for something that you want to be truly portable, that you want to give to other users. If you embed the object, then a copy of the object exists both in the original document and as an embedded reference. So, you have duplication of the object, which will take more disk space. But, you have a totally portable document as well.

The second area I wanted to discuss is DLLs. DLLs are simply a collection of compiled routines. They run the gambit from simple utilities to full-blown applications. The key to DLLs is that they do not need to be available at compile time for a program. They only need to be available at run time, thus, the dynamic. Some examples of this would include the equation editor provided with Microsoft's word-processing product. It's a third-party product, implemented as a DLL. Since it's implemented as a DLL, it's available in their word-processing software, the editors that come with Windows. It's available from spreadsheets. It's available in graphics packages, any place that can serve as an OLE server because this is actually an instance of something that is both OLE as well as a DLL.

DLLs are often compared to source-code libraries, the difference being with source-code libraries you have a single copy of the source code but multiple instances of it when it actually is compiled in multiple programs. With DLLs you have a single copy of the object code, which is then loaded at run time, so that you actually only have one copy of the object code laying around, which conserves disk space and memory at execution time because the DLLs are intelligent in the way they're loaded. You never had a second copy of the DLL running in memory. If you have multiple applications accessing the same DLL, they all use the same code.

There's a new standard coming with Windows NT that creates some more flexibility in the DLLs. Currently they're rather limited in the data they can contain because of this common-code segment. The new standard will allow a data portion with each application or instance of the DLL, but still maintain the common code, which I think will increase their useability and flexibility even more.

Another use with DLLs during this transition from DOS-based to object-oriented systems is that often a calculation engine can be developed as a DLL and then the interface developed in both a DOS and a Windows version. Rather than a source-code library that gets compiled into both versions, you can have a DLL that can be loaded at run time. That's just been made possible recently with some of the compilers that now are bundled with DOS extenders that allow them to run the DLLs from DOS. DLLs currently aren't executed from DOS without some sort of management program running them.

Another example might be some of Larry's rate routines as he moves to object-oriented programming. I'm not familiar with their structure, but that's something else that might be able to be implemented as a DLL, and if they're used in multiple

occasions, you'd reduce the number of instances of the object code and have something that took fewer resources when it actually came to run time.

One of the benefits of DLLs is that you can use any available language that suits your purpose. Currently getting programs written in different languages to talk to each other, while certainly possible, is a bit tedious. When you're working with DLLs, that's all handled at the operating system level, rather than at the application level. So, if you have pieces of code that are written in Basic, Pascal, Fortran, C, C++, as long as they're DLL compliant, they can all talk to each other rather seamlessly.

This leads me into my example. The example I'm going to use is one of Regulation 126 cash-flow testing. The document that you want to produce is probably best suited to a word-processing document, although much of your data may be numeric in nature. But, I think the way I would start it out would be to rough out an outline of the document. I'll assume that the calculations software I've used has created a database of output in some standard database format, Dbase or Paradox. I would then use a third-party DLL report writer to access the database and create proforma financials and some nicely formatted reports.

There may be some areas where we've done additional manipulations and spreadsheets. Use OLE to open up a spreadsheet display, input variable assumptions. There might be another OLE object in there that's again, a spreadsheet displaying results from the assumptions. It could be graphic support, which would again be an OLE object that shows graphically some of the results. You may document some of these using an equation editor, which could be actually an OLE object within either of the spreadsheets, the graphics package or just from the word processor itself.

As I said in my beginning, I think we've reached a critical mass in the marketplace for object-oriented operating systems, and we're starting to see more development using the tools available to integrate software that's available in object-oriented operating systems. It gives us a seamless integration. It shifts our focus from the tools available to the result desired and allows us to use the best tool available for the job. I think these benefits will greatly increase the productivity of actuaries using object-oriented operating systems.

MR. GEORGE L. ENGEL: Larry, I have a question for you. One of our problems is access to programmers because of them being in a separate IS department and having other projects for other people, etc. At a former company we had a system similar to what you have. One of the problems we had, though, was that we had skilled professional programmers who felt like they weren't really part of IS, and they weren't really part of the actuarial department and we had a high rate of turnover because of that or at least that's what we think it was attributable to. I'm wondering if you had any of that experience or if you have some kind of career path for these people, or what you do to help keep people in the department?

MR. CURRAN: That's a good question. We actually have had very good experience with turnover in the last several years. We've had no one leave. We do have a career path that will take a person up through the equivalent level that we will have a recent FSA or new FSA go. After that, the ASD career path really is closely tied with the IS department. We use an identical career path up to that point, the same as the

IS department does. We use the same titles. Job descriptions vary a little bit. But, our ties are so close we actually work with the IS department. When we do a promotion, we run it by the IS people to see if it matches how they would do it. So far we've had good luck along these lines, and we think with the close ties with IS – we've had several invitations for our top people to go over to IS and we at some point will allow this to happen so that they will have a career path leading through officer – that we're pretty well set.

MR. PAUL A. CAMPBELL: Larry, I would like to ask you a question and the other two could comment as well. Are you aware at this time following the origination of your ASD that a lot of companies in the industry have followed the same format, or do you feel that yours is still somewhat unique in it's structure? Second, how would you suggest that computer science education for undergraduate students can make sure they will fit neatly into today and tomorrow in the systems?

MR. CURRAN: I'm not aware of what other companies are doing in this regard. My general impression, though, is that it isn't enough because I hear a lot of comments on both sides. I've heard some comments here about the difficulty of working with IS. A few years ago I attended a Life Office Management Association convention. That was really primarily for IS people. Someone was, at that point, talking to me about the rate routines and showing a great deal of interest, and all of a sudden the person found out I was in the actuarial department and he literally turned around and walked away. Obviously I couldn't know anything being in an actuarial department.

In terms of training, the things that we're really looking for now is to have the universities provide training in object-oriented technology. We're more interested in that at this point than I think anything else.

MR. MCANDREWS: My rather limited experience has been that the set up that Larry described is unusual, if not unique. Of the companies I'm familiar with, none of them have this particular structure. It's a much more traditional structure, IS on one side, the actuaries on the other.

MR. BRYN T. DOUDS: I have a couple comments on this topic about should we have a systems staff in the actuarial department. We tended to always be very separate because of the control issues. You're going to have one staff develop a program and a different staff under a different manager test it. It's kind of like dotting the numbers. Although recently we've begun working with APL, and we've found that none of our systems professionals would touch it because there's no career path fundamentally. So, we now have some of our own programmers, and we actually took actuarial type of staff with a strong math background, and they're at least willing to work with the APL. Could you comment on that a little further?

MR. CURRAN: Well, actually our actuarial students do work with APL, but about the only systems they work closely with are the pricing programs. In terms of for valuations, being larger systems, we felt that the traditional languages were more appropriate. We found that the actuarial students did not care normally for rotation stints within the systems division. First of all, we required a two-year rotation to get the benefit out of it, and they felt that they were losing track of where they were going at that point. So, we gave up on that and just strictly brought in math people.

One thing that we have been very careful about is, we try to bring people in the systems division who do not have an interest in becoming an actuary. We don't want it to be a back doorway of getting into the actuarial career path. The testing that we do in the ASD is basically the same as IS. It's as though we were in the IS department and physically located somewhere else.

MR. ROLAND R. ROSE: I have a question for Larry. I'm curious about how the duties are split in your development and maintenance of illustration systems?

MR. CURRAN: Well, that's an interesting question. Originally the sales illustration systems were done by the ASD within the actuarial department. With the advent of the PCs it was decided that the real PC expertise was in IS, and so the sales illustrations moved over there and took about half of the then systems division with them. But, part of the rather complex agreement is that all testing of sales illustration systems would remain within the actuarial department, and we actually maintained some programs that would basically calculate sales illustrations that we would use to do the testing. Things are kind of evolving. Much of the calculation for sales illustration is shifting back into the actuarial department at this point. In fact, there's a move pretty much all over the spectrum for calculations that have to do with values or cash value or anything to now done in the actuarial department in multiple-use routines. But, the sale illustrations are done in IS. Checking is done in actuarial.

MR. THOMAS L. BAKOS: I wonder if Matt could get Larry off the line here. You commented on testing various different kinds of equipment. Do you test PCs other than the IBM standard PC? Do you have any comment on clones?

MR. TUCKER: The standard PC that was recommended within our firm by the central computing resource group is an IBM PC. We have about 25 PCs in our office. We have one IBM PC. We have used clones since day two, not necessarily day one in our group, and we have a wide variety of clones. We have Dells, Gateways, some brands you've never heard of, I'm sure, Compaqs, Everex, all different sorts of brands, some of which are maybe not in business any more. That's one of the dangers of the clones. We've actually only run into a problem once with a clone, and that was a clone that was a no-name brand. The company took it back, actually. There was a problem with the mother board. But, we've had excellent experience with clones. We do run a Novell Network, and we've had no difficulty with putting them on a network, even though central computing said it wouldn't support any of our questions. So, we didn't have any.

MR. EDWIN R. SCHRUM: I just wanted to comment on that audit concern about APL. We've had no problems with our APL, which we're heavily involved in. We run all our program changes and our pricing model – which is deemed to affect profitability -- through a second APL program. That seems to be satisfactory. I'd hate to think anybody would avoid APL just for audit concerns because our auditors are very comfortable with that.

MR. ROY MURPHY: I've got a comment that I'd like to make about a fairly new product category that has come up lately, and that's the whole concept of multidimensional spreadsheets and how they are as modeling tools. I've been using one particular package for about a year and have managed to replace practically all of

the traditional spreadsheet work that I do with this package called CA Compete from Computer Associates. I just wanted to mention what I thought some of the advantages of this kind of thing were and maybe get some other people in the profession thinking about doing work like this.

Lots of times we're doing work with sales that are defined by many different variables. You can place each one of those in a dimension and then reflect a lot of different factors of the data you want to model in one place and then build fairly English-like formulas that refer to things by the item name rather than by cell references. It makes the work easier to document, easier to check and understand, and the products make it easier to change the reporting. So, from the same data you can show reports that are actual versus planned or show the same data flipped around another way to show a series of things over time. I really would encourage anyone here who's got a little bit of money in their software budget to maybe spend $99 on Lotus Improv, which is available for a limited time at that price, and keep an eye out for an upcoming version of CA Compete, which is supposed to advance its capabilities. Just give this product category a chance and give it a try.

MR. STROMMEN: I might make one comment on that. At Northwestern Mutual, the marketing department people recently discovered Lotus Improv, and they're putting a lot of their sales data by product, by age, by general agent and that kind of thing into that kind of spreadsheet format so they can flip it and look at it, and summarize it any which way. They're very enthusiastic about it.

MR. CHARLES ROBERT DOLEZAL: You talked quite a bit about actuarial applications. Our company is in the process of picking a new administration system, and I'd like you to comment on some of the PC-LAN-based systems that are out there.

MR. MCANDREWS: I think the best response I can give you is a quote from Mark Twain, and that's "When all else fails tell the truth." I really don't have any exposure to that.

MR. STROMMEN: Is there anybody in the audience with exposure to LAN-based administrative systems who might be able to help him out? If not, I have another question. The speed and capacity of PCs seems to be doubling every year, and actuaries always seem to want the high-end PCs. As a practical matter in your organizations how often do you find that people are upgrading their hardware?

MR. TUCKER: Every day. Basically, we have a wide variety of PCs that we still use all the way from 386 20 megahertz machines. We might have some 16 megahertz machines up to 486 DX2/66 and that's basically what we do. We move the machines around as we get newer machines. But, basically we upgrade as we see fit and as budget allows. We don't usually go out and buy the newest one that's hot off the shelf because the price is usually jacked up pretty high. But, we don't wait too long either before we get into the newer and faster PCs.

MR. CURRAN: It is basically the same thing here. We get our computers, however, through the company store, and we upgrade through there. We tend to make sure that the people with the most critical need have the fastest machines, but they're generally not the state-of-the-art. However, within our area now we do have

probably 10 to 15 Dell 486 model 50s. We use those in the systems division for some of the actuarial students who are doing some intensive pricing where they're running into time problems. Pretty much, though, whenever we hit a situation where we can justify it, we get it, whether we budget it or not. So, our luck has been pretty good.

MR. SCHRUM: I just wanted to comment on your strategy of how often you replace machines. Our company has a strategy in the individual actuarial that we buy state-of-the-art machines. Right now we're buying the 486/66. We go on a pass-down principle: our biggest programming students will use it for only one year. With next year's budget, we'll buy some Pentiums for those students because it seems, as the machines get better, we always seem to be pushing them. We pass it down within our department for the second year, and maybe a third year. Right now I can say that we have 286 machines that I purchased five years ago that customer service thinks are the best word processors they ever had. I literally swiped their $500 a year computer budget by handing them what I consider junk. If I pass around my junk, I can justify getting what I call real equipment.