RECORD, Volume 28, No. 2*

San Francisco Spring Meeting
June 24–26, 2002

Session 71PD
Hot Technologies

Track: Computer Science

Moderator: WILLIAM FRED FARRIS
Panelists: JOSEPH ALAIMO
RICHARD A. DERRIG†
MARK A. TILLMAN

Summary: Technology is regularly used to leverage insurance and pension data to accomplish some ordinary company operations (underwriting, claim settlement, pension plan administration, etc.) in a more effective and efficient way. This technology may be sophisticated or not, but it involves lots of data and lots of processing that actuaries need to understand, at least conceptually, because it is expensive.

MR. WILLIAM FRED FARRIS: In this session, we will discuss three different aspects of computer technology. Dr. Richard A. Derrig will talk about ways to detect false claims. Joe Alaimo will talk about programming for virtual machine and runtime environments. Mark Tillman will talk about the use of a computer program as a communication tool. Our first speaker, Mark Tillman, is an Associate of the Society of Actuaries. He earned a B.A. in mathematics from Cornell University. Mark is a managing director at Winklevoss Technologies, LLC, also known as WinTech. He joined WinTech in 1994 as a developer of ProVal evaluation and forecasting systems for pension and health actuaries and business consultants. In addition, being directly responsible for the actuarial engines within ProVal, he directs the research and development of the entire product. Mark also oversees the customer support and training of ProVal clients.

MR. MARK TILLMAN: Mark Ruloff, who was supposed to have given this presentation, actually wrote this speech. He wasn't able to be here today, and one of the main differences if he had been here is that he would just take you right through this speech. But I am going to inundate you with a few quotes from Alan Cooper.

†Mr. Richard A. Derrig, not a member of the sponsoring organizations, is senior vice president at Automobile Insurers Bureau of Massachusetts in Boston, MA.

Alan Cooper's writings are delightful, unlike many of the technology books that you read. This guy has attitude, spunk and a sense of humor. I highly recommend his books. There are two that I know of. One is called *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity,* and he has another book called *About Face: The Essentials of User Interface Design.*

I hope today to share with you some of the lessons that we've learned recently with regard to software design that may be useful to you, whether you are involved in designing software products, have input into the software products, or are involved in looking for products, whether you're buying or leasing software. I'll tell you about some of the things to look for when designing software or buying some of the packages.

We've had a product for a number of years that is designed for actuaries. We're actuaries, we write products for actuaries, and we've been able to go along with our gut instincts. When we came to a new product for outsiders, whom we'll call external users—specifically the consultant, who may or may not be an actuary, or the pension plan sponsor—we had to do some more research and think about the design. That's where we're coming from today. We'll contrast it with what you may be familiar with right now—the systems that you use internally as actuaries—and discuss some of the considerations you might have in trying to reach a wider audience.

I think our company's history is similar to some of your firms' histories. We started out with computer systems because the actuarial calculations are so intensive. It's just a natural. Of course, you have a computer system. The early ones were just code. As they progressed, eventually they had some user interfaces. At some point, the user interfaces seemed to make sense to the actuaries, so they said, "I think anybody would appreciate this system. Anybody who's interested in knowing what the liabilities are or knowing what the contributions or expense numbers are would be interested in using this system."

So you or some of your colleagues may have taken your internal software and tried to use it in front of a pension plan sponsor or actually said, "Here, let me share my software with you and have you play around with a few things."

We tried it; in fact, we thought that every pension plan sponsor on the planet would want to use it. Ten years ago, when we started the project, we thought it would be so easy. It turns out it isn't that easy. Plan sponsors don't want to be actuaries. That was the first lesson that we learned. Some firms, some of the biggest firms, have even declared that a dry market. There's no way that plan sponsors would ever want to use the software.

We believe, however, that there is a market. It's just that the earlier attempts failed in the design. The problem with trying to reach the plan sponsors was taking an actuarial system designed for internal users and just throwing it to the plan sponsor or to the consultant.

I think it's not unlike—I told you I was going to quote Alan Cooper, so here we go—if you remember about 10 years ago in 1990, the PCs were coming out, laptops, and we were all trying to go paperless. There was a company called GO Corp. that came out with a computer called PenPoint, and it was supposed to be a handheld computer, as the progenitor of the next generation of handhelds. It took about two years, but it crashed and burned. It was just a disaster.

At that point, Apple came out with the Apple Newton. No one had heard of GO, but Apple would be the company to make it happen. Well, that one crashed and burned too. There was another casualty along the way. The next year, General Magic came out with the Magic Link. It crashed and burned.

It wasn't until 1996 that this little guy came out—the Palm Pilot. I don't know how many people are using them these days. The market is always ready for a well-designed product that does its job well. In fact, if you look at the Palm, it did less than the earlier predecessors. It just happened to be well designed, did what it was supposed to do, and met the goals of its users.

Speaking of users, what we'll do here is try and meet those external users' needs, but let's relate it back to what we already know and the people who are already using actuarial systems.

In the pension world, we have four main characters. We have the analyst, and there are some tasks that the analyst does. But I want to focus on who the analyst is. Analysts, generally speaking, are kids right out of school—we were all there at one time. They've got zero to five years of experience. Maybe they just went through an internal training program on what a pension plan is, what an interest rate is, and what v to the t means and tPx. That's about where they're coming from. They're using the system to produce the initial numbers, and then those numbers are reviewed by an actuary.

By actuary, I mean someone who is an ASA, an EA or maybe even an FSA. He or she probably has five to 10 years of experience unless he or she has decided to just be the technical actuary. The actuary may actually have 30 years of experience. We're not talking about the client relationship manager here. We're talking about the technical actuary who reviews the work of the analyst and then prepares the report.

After that work is done, there may be an investment advisor who, in most cases, is not an actuary, but someone who is familiar with investments. The investment advisor probably specialize in institutional investing or may specialize even further into defined benefit investments. For the most part, they just know that there's funding and there's accounting, there are contributions and expense. That's about the breadth of their knowledge. We need to meet their goals, which are forecasting the contributions for the plan. Finally, what we'll focus in on today are the consultant or pension plan sponsor and the needs that they have.

Speaking of needs, we have goals, and we often mistake goals for tasks. To differentiate a goal from a task, imagine you're here on Saturday. Actually, you're home on Saturday, and your goal is to relax in the hammock that you've just put out between the two nice oak trees in your backyard. That, however, is not your task. Your task is set out by the head of the household, or maybe by you just so you have a clear conscience, to mow the lawn underneath the hammock. The task is simply a means to an end, the end being your goal to relax.

For software, these are universal goals that are out there. We have personal goals, and we have corporate goals. Everyone's personal goals are basically the same. Number one, don't feel stupid. Software should not make people feel stupid. The software should not encourage you to make mistakes. It should allow you to get an adequate amount of work done, so you have some dignity, some pride, a feeling of accomplishment. You should have some fun. Or at least you shouldn't be bored.

To understand not feeling stupid, has anybody purchased a DVD player, a VCR or a new television in the past few years? How long did it take you to get that thing up and running? Usually it takes two or three hours and involves some head banging against the manual. Software should be designed the same way those things are—with the concept of the principle of commensurate effort.

With the television or the DVD player, as soon as humanly or mechanically possible, that thing should be able to be plugged in and give you a picture. If you want to put more effort into it to use the new features or any of the other fancier stuff, you put more effort into it and you'll be rewarded. The same should be true of software. You should get some kind of payback as soon as possible.

Unfortunately, software and many mechanical or technological devices have not followed that. The manufacturers always have said, "Well, we'll just put it in the help, or we'll just put it in the manual," and that's not the solution. The software or the interface should be designed for the function.

Those are our personal goals. Our corporate goals are also universal. Whether it's your actuarial firm or your client, you want to increase profits, increase market share and defeat the competition. You may want to hire more people, offer more products or services and build up the business. Now, the bridge between those two comes into practical goals, whether it's your firm that wants to produce evaluations on budgets, avoid mistakes, that kind of thing. We're going to concentrate today on the external users whose goals might be to minimize the contributions and expense, to avoid surprises and offer competitive benefits.

They might have a number of other goals as well. We'll take the consultant and the plan sponsor to be one, to have the same goals, with the consultant's really being an extension of the plan sponsor's.

How does this all work beyond the nice, comfortable framework of our offices? Well, the consultant and the plan sponsor get together, they discuss the results that you've come up with, they ask some questions, and then you go back to your office and work through them. We'll talk about that here in a second as we get through the design of the software. But that's the overall framework.

Let's see how the software is going to help us do that. We have our analysts, our actuary, possibly an investment advisor and the consultant. If we go back to the analysts, their job is to get the numbers in and the results out. So that you'll have a concrete example of what I'm talking about, I'll demonstrate using my company's evaluation system so you can see what I mean.

For the analysts, in this particular product, it's a three-step process. You set up the inputs on the input menu, you run the results on the execute menu, and you take a look at the output on the output menu. The analysts are basically living in the input area. They're going to set up the benefits of the plan. So we go into the plan benefits, and they set this one up with three different benefits: the death benefit, the retirement benefit and a termination benefit.

They further define what the retirement benefit was, what the benefit formula of that retirement benefit was and so on. They put a lot of work into this. There are places for them to enter the valuation assumptions, whether they be funding or accounting. They have slots to put in all of the interest rates, salary scale and trend rates for your retired medical claims.

When the information is entered, their goal is to get it to their boss and look like heroes as soon as possible. What they're going to do is go to the execute function and run the valuation in the sample life and take a look at all the output, quickly run the sample life, and then make sure by checking the sample lives that everything seems to be as they intended. That's how the analyst works.

Next comes the actuary to review the data. The hard way to do the actuary's job is to go through exactly the same steps that the analyst did using exactly the same screens. The software isn't really helping you if that's what it makes you do. There should be an easier way, a design for the actuary. What the actuary wants is to go to the end product, which the analyst has completed, and view the results, both the output and the input.

If we go to the input, the software allows the actuary, who is reviewing all these data, to take a look not only at the results, but also at all the work that the analyst did. Out system provides a table of contents of all the work that the analyst has actually done, and the actuary may come into the system or just ask the analyst to print the information out to be reviewed on the plane, train or wherever the actuary happens to be.

The process is similar for the investment advisors, who don't want to know what the actuary has done other than that it's right. They don't want to project the contributions and expense, and for the investment advisor to bypass this, they're going to have their own screens that build on what the actuary and the analyst have done.

Now we'll see input, execute and output functions. They'll go to the deterministic assumptions. They're interested in the investment return, so they'll come in here and say, "Well, let's say if the investment return was instead of 9 percent based on the past couple of years, maybe we'll assume - 20 and then - 10 and then zero. We'll assume everything else is the same."

They've got their own screen, their own way to interact with this to meet their goals as quickly as possible. Say they have a screen called bad news, they can execute their deterministic forecast simply by switching in the baseline assumptions that we just created to bad news. If they run that, they get what they want, which is that they can take bad news, line it up against that baseline with the actuary there, and see that forecast contribution for all or any years that they're interested in.

The last step is that the consultant of the plan sponsor wants to take a look at all these results. More importantly, they want to understand the results and want to make decisions from the results. The way that's normally done is to have a consultant act as the intermediary. This may or may not be an actuary. It could be that the client relationship manager would rely on the technical actuary to translate the numbers, but the consultant wants to present the results in a big-picture way, so he might take those results and graph them and paste them into a Microsoft PowerPoint presentation or Word document.

The problem with this, say, the PowerPoint slides, is that this works fine to a point. This allows you to present results that you have determined, go to the plan sponsor, let them ask questions, let them know the answers to any one of those questions that you have anticipated, and usually you'll have plenty of questions that you hadn't anticipated. You go back to the office, and about a month later come back to the results and then decide on some kind of actions. This is the point where we thought that there really ought to be a market for a piece of software to help with this process.

The fact that I have to stand up here and say that you go to your clients and tell them, "I don't know the answer, I'll come back to you in a week or a month," makes it seems like there's an opportunity for improvement. There's got to be something that this software can do. So what can we do?

We thought, "Let's replace the static actuary report that we've all been writing for 50 years or more on paper. Let's replace the PowerPoint presentations, which people throw up. They're great but lack that dynamic quality. Let's create the electronic actuary report or dynamic PowerPoint presentation," whatever you want to call it. Let's say that you get in front of your clients, and they ask you, "What if I were to change my interest discount rate?" Everyone is excited about interest discount because—I don't have to tell you that rates are low. The reason you have to say, "Well, let me go back to my office and do that," is you don't want to show the client that this is what you have to do. Go back to input valuation assumptions, go to your valuation assumptions, change your interest rate, grab your interest rate, change your new interest rate, go back to your valuation and rerun your valuation. There are many steps.

That's the whole point. This is what I wanted to show you. You know how this works. Run your valuation. When the software works, it needs to be flexible enough for the actuaries, and it is. We save this and recalculate contributions and expense. I think we probably made the point. All we wanted to do was change the interest rate. What if there were a dynamic system that would allow you to do the same thing with the click of a button?

Here it is—a few tools to replace your actuary report, or at least augment your actuary report. With the click of a button, we're going to change that interest discount rate. You can see the impact on expense, you can see the relationship to the baseline that you had shown, and you can answer questions on the fly. You can use this as a presentation tool in front of your clients, or you can see use as a leave-behind, so that clients can start to generate questions on their own and not have to go back to you. You could also allow the plan sponsor to give it to other interested parties. Maybe you want to give it to the auditor to play around with. The sky's the limit.

In designing the software, there were issues that we had to figure out. Hindsight is 20/20, and I wish we known this going in, but one of the issues was that the plan sponsors didn't want to be actuaries. They didn't know where to begin. There's a lot of stuff that didn't follow the rule of the principle of commensurate effort. It took two to three hours to plug in the DVD player. You didn't know where to begin, and finally, you gave up and read the manual.

There are a lot of complex parameters. How are you going to deal with all that? You know the DVD player has all these fancy options. How are you going to deal with all those things and let people work this easily? You don't want them to accidentally damage any information. There's some information that plan sponsors aren't qualified to handle. You don't want to let them damage it, and you might not want to show them some internal information.

Maybe you're loading for the Retirement Equity Act death benefit, or maybe you're ignoring the 1987 grandfather benefit. There might be a million perfectly reasonable, wonderful details that aren't worth getting into with the plan sponsors. How do you deal with these? First, start at the end and respond to questions. That is, get the system up and running as soon as possible. If we show examples of that here, the results are already done. The end product has already been done. You can simply modify the end result to get what you want as we did here by clicking around with the discount rate.

For the complex parameters, that's easy. Again, we're starting with the end result. Another way to put it is that the internal user, the actuary, has already set those up. To shield the plan sponsors from all the complexities, we'll just give them a limited view. There are hundreds of actuarial assumptions. Each actuarial report has at least five pages on actuarial assumptions. You see only seven or eight of them right here. Know the mortality assumptions, down to the complex things. Just show the primary points of interest.

Another lesson we've learned is, if you don't want people to find something in your software, bury it. It's easy, and they'll never find it. It follows the principle of commensurate effort again, which is get the system up and running. If they want to exert more effort and find the hidden information, they'll find it.

What do I mean by that? Take another tool that's in here—the asset allocation tool. Here's a comparison of contributions at different percentiles. It has four different mixes, so if the plan sponsors want to look at the current mix versus the conservative mix versus the average mix versus an aggressive mix, they certainly can do that and see how the pension plan performs. It's not limited to that, though. It just so happens that we buried under a button labeled "set mixes" the ability to put in more mixes. So you can pick up mixes from the efficient frontier. I can double-click there and add another one called EF 101 (efficient frontier 101), so now we have five mixes. But the fact that it's buried means that you don't have to deal with it right away, and believe me, people will not find it until they're looking for it.

It's the same principle and same solution to a different problem for all those issues where you're afraid of the clients accidentally damaging information, or you don't want to show internal information easily. Don't show it to them. The plan sponsor is interested in more of the big picture, and the interface on that kind of system for those external users, the nonactuaries, is going to be a focus on the big picture. That means a lot more graphs primarily.

It also could mean a customizable interface. If they want to get some of the technology navigation stuff out of the way, there'll be options in here, if you want to turn up tabs on the right-hand side and focus in on a graph, you can do that. And if they're interested in presenting this to their board members, you can fine-tune a chart with various options.

To review, you start at the end: you view results that are already available and change them by clicking around. The result is that clients and consultants have better communication. By giving you more power, you have an interactive presentation that is not just static, the presentation could be more interesting, your message will come across better, and more timely decisions can be made.

The last thing I want to leave you with is that as soon as you go external with your software, there are a number of different issues that you have to deal with, including security and software updates. You can't just walk down the hall and install it anymore.

MR. FARRIS: Joe Alaimo is the chief architect at ProComp Consulting and specializes in actuarial software development. Joe has been with ProComp since its inception in 1985 and has worked on the development of a variety of insurance systems, including illustration systems, needs analysis and a number of insurance concepts. His involvement includes not only the architecture and development of calculation engines, but also developing usable user interfaces and industry-compliant reports.

He has developed systems for a wide range of insurance products, including term, perm (par and nonpar), disability and critical illness, with a special emphasis on Universal Life calculation engines and systems. Joe has been involved in and is a firm believer in the development of user interfaces that are usable by people other than actuaries. Joe has worked with a variety of development languages, including Visual Basic, C++, APL, Delphi, and .NET (pronounced dot-NET).

MR. JOSEPH ALAIMO: Thanks very much. Today I'm going to talk about the Microsoft .NET technology. I will try and stay as nontechnical as I can, but I can go only so far in that vein, so bear with me.

Let's start by discussing what .NET is. I'm making a personal leap here. I think that it is a concept that will revolutionize software development and take over within five years. That's a pretty bold statement. But once I finish this discussion, you might see why I'm saying that. Right now Microsoft is promoting .NET with advertising, calling it one degree of separation. You may have seen its advertising. I think the company is taking a cue from McDonald's with McChicken and McThis and McThat. It's .NET this and dot that. As more time goes on, you'll be hearing a lot about .NET from Microsoft.

Basically, it's a virtual machine and runtime environment intended for hosting more then just a single language. I'll get into more of these points, so if you don't understand that yet, that's fine. Its point is to give easier access to Internet development and easier development of Web services. It's basically a way of programming the Web. Right now, most Internet programming involves multiple languages, embedding scripts within HTML pages and Perl scripts and lots of different ways of getting something, but not real programming. Microsoft's solution is to allow a program to be written. Java does allow that right now as well.

Now, what is the .NET process? How does one go about developing a .NET application? Microsoft in February released its Visual Studio .NET, a fully integrated development environment that lets you drag and drop controls and put your code in. Source code is generated inside the .NET programming environment in any of the .NET-compliant languages. This source code is then compiled into the Microsoft intermediate language (MSIL) as an executable (EXE) or dynamic link library (DLL). It's not a full, true compile; it is an intermediate compile or a P-code compile.

When the application is run for the first time, it is then compiled just in time (JIT) into a managed native code for that particular machine. What that means is that it then gets compiled into full machine code at that time, and then the virtual machine or common language runtime control program runs the application. One of the upsides with this, although Microsoft obviously will not develop for anything other then Microsoft machines, is that another company could in theory develop a JIT compiler for platforms other than Microsoft platforms for these languages, much like Java does today.

I'll probably make a lot of references to Java because I think Microsoft started the whole .NET initiative to come up with a better Java. Microsoft's purpose, I guess from a marketing point of view, was to usurp Java's foothold in the industry. The .NET process that I just outlined for you is this. You start with your source code and compile into MSIL, which has type verification. The execution has type verification, your JIT compiler compiles into managed native code, and then your common language runtime (CLR) will run that EXE.

What do I mean by multilanguage? Microsoft has published the specifications for the programming environment and the MSIL. What that means is that any language can be used in .NET, as long as it complies with these specifications. Any third-party company can now take a language that you might be familiar with as long as it compiles the language down to the same MSIL or intermediate language, which then can be run in .NET.

Many popular languages right now already can convert to .NET. There are 30 to 40 companies who develop languages that comply with this standard. What that lets us do is write a procedure in one of the .NET languages within that environment, for example, VB.NET (Visual Basic dot net), and I can call a procedure or code that's written in a different language, perhaps C#.NET (C sharp dot net), and I can write within my environment, step through my code, call the other procedure, step into that other code and see exactly what's going on line by line. When it's done, it will convert back to my code.

This allows multiple people on a project to leverage the knowledge they already have. If you have Java developer, a C# developer, a C++ developer and a Visual Basic (VB) developer, you can break out that program into different pieces, and everyone can program in the language he's most familiar with. Retraining people might not be as extensive as it would be if Microsoft created just one language and forced everybody to learn that one language. Microsoft also includes a set of extensive class libraries that comprise practically any functionality you could ask for, such as reading and writing to disk and your common sets of functionality in libraries.

This encourages third-party vendors to create .NET versions of different languages. More than 30 languages have already been ported because Microsoft has had .NET out in beta for about a year-and-a-half and have had input from a lot of people working with such languages as Pascal, COBOL, Java and our favorite, APL.

For legacy systems integration, multiple language support can be a lifesaver. If you have an old COBOL program that you want to bring up, and you have one guy who knows COBOL, he doesn't have to relearn C# to get it up there. He can use COBOL, you buy the COBOL.NET compiler, and he can use existing code, which he might have to tweak to fit into any of the new functions, but the language itself is basically unchanged. In theory, people who write these compilers might allow you to take that COBOL code and not change it all and have it running under Windows.

The .NET languages themselves are all fully object-oriented and support visual and object inheritance. For people who use VB, as an example, up until now VB hasn't had that capability, so any language has had to comply to those specifications. Components can be bundled into packages that can be marked with versions, thus avoiding DLL hell.

In the past or in the current environment, Microsoft has taken the approach that you can have one DLL or one ActiveX component, and only one with that name can be registered on your system. You may have had the experience in the past where you upgrade a component in your program, and another program stops working because it has a different version of a DLL, and both versions have the same name. That's known as DLL hell, because once you get in there, there are times when you just can't escape from that. You have to choose to run Program A or Program B, but you may never be able to run them both. .NET avoids that by using version numbers, so I can have two versions of the same DLL, and, say, Program A uses version 1, and Program B uses version 2.

One thing I want to say about the .NET language is that when Microsoft created the .NET language as opposed to just taking an existing language, moving it up and bringing along all that baggage, it started with a clean slate. Microsoft decided to develop a language much like Sun did with Java and take away any preconceived notions. The developers asked, "If we could have anything in language we wanted, what would we put in there?" They started with a blank slate and built it up from nothing. I think that a lot of the problems in previous languages have been removed in .NET.

By having a common runtime, this intermediate language allows Microsoft to offer implementation over a range of hardware and software architectures, including handheld devices such as PDAs, cell phones, Palm Pilots, embedded systems, DVD players, microwaves, that kind of thing. I think that's really the goal in making a platform-independent system. As I said, that does open the door for someone to create JIT compilers and runtime engines for platforms other than Windows. I don't think you'll see Microsoft making any UNIX compilers, but it's open for someone else to do that.

This allows the same backend core components to work on desktop and Internet server machines. That's another important point. I'll get into the Internet stuff, but you can develop your backend components in Microsoft .NET technology, and then your front-end components could be one for your desktop and one for your Internet application, but the backend code doesn't have to be changed. You can run the same code on both the Internet and desktop.

The CLR has an architecture that provides several advantages. There's less concern with internal plumbing; it has expansive tool support and simpler deployment. Again, it offers versioning, so you're not in DLL hell. It has superior scalability, support for multiple programming languages and a common data type system.

I'll tell you about the common type system because every language has to have one set of rules it has to follow. An example I'll give you is that a C++ integer equals a VB long integer. If you declare a variable as an integer in VB and an integer in C++, they actually mean two different things. C++ uses a four-byte integer; VB uses just a two-byte integer. That kind of consistency or lack of consistency now goes away. Every language has the same data types. Integer means integer everywhere. You know you float double all those things. The nice thing with this is if you're ever converting data from a C# application to a VB .NET application, or any .NET application, consistency makes these conversions' data transfer a lot easier. Whether you are writing two or four bytes to a file is no longer a concern.

.NET has a framework of classes that are available in the .NET language. It's quite comprehensive and is available to all the languages in the .NET framework. The function calls to these classes are identical for all of the .NET languages. If you're writing in APL.NET and want to write code to a file, and there's a function called save to file, if you then switch and write the code in VB.NET instead, the function is still called save to file. If you switch to C#, it's called save to file. Following the common framework of classes reduces training time in the event that you have to switch from one language to another, switch your whole team, or people want to upgrade a little bit here and there.

Also, if you do have people developing in APL.NET, VB.NET and C#.NET, it is easier to understand the next guy's code, because even though you may not the greatest in APL, you'll know save to file means save to file. You know what it's going to do. It's not going to surprise you. I think that, overall, .NET gives your code a greater consistency, and for long-term maintainability, that's an important feature.

What is the net in .NET? .NET implies Internet, so what is the .NET? The .NET takes two forms. In .Net applications, there are Web forms and Web services. I'm not sure if you're familiar with either of these, but I'll go through them briefly just go give you an understanding of what they are and how powerful they can be.

A Web form is what you see on a desktop. When you open an application, you see a form with a lot of input buttons, controls and "okay" buttons. That's basically what a Web form is, but on the Web. What has been done in the .NET platform is you can develop Web forms using the same drag-and-drop interface as is now being used for desktop applications. If I want to draw a Web form, I just have to open up my Visual Studio .NET, put a form on the screen, grab an input box from the toolbox, drag it on and place it on my form. That's where it's going to stay, and when I run this on the Internet, that's where my users will see it.

For the Internet application and the desktop application, drawing the forms is different because the Web is not going to offer as much functionality to your forms as Windows can. If you're calling business rules as an example, you separate all your business rules outside of the form itself, and you have different classes and different code. Then you can use that same code in your desktop application and your Internet application.

That's what we're doing right now. We're developing Internet forms as well as desktop forms, and we're developing the architecture so that all the business rules and all the insurance logic are staying outside the form itself. All we have to do is redevelop the form for the Internet. We don't have to redo the whole application.

For a Web form, all the processing is performed on the Web server, and then the display information is passed to the browser. This reduces the need for software in the client's machine. Right now when using Java applets, you have to download an applet and run it on your machine to do all that processing. Microsoft is taking a different tack to it. Now all that information stays on the server, which does all the heavy work. It's hopefully a very powerful machine, so it does the work and sends the results to your browser. A negative effect is that it potentially this result in an increase in traffic across the network.

In another session, I mentioned Active Server Page (ASP), and not everyone knew what that was. ASP is a way to develop a Web form whereby you're embedding script inside your HTML page. When users click on this button "Run the script," the script does these things. This puts your script right inside HTML. But now ASP .NET means that you're using these WinForms with a .NET-compliant languages to compile the application and run it on your server. Now it's actually a true compiled program.

What's a Web service? A Web service is another technology, not started by Microsoft, but it comprises small, discrete building-block applications, each handling a limited set of tasks that connect to each other. You write a program that you put on your server, and there's an interface available for other programs, whether it's from your own company or from outside sources to actually hook in to your service and run your program to get results.

A service is accessible by the general public, but a username and password may be required. You can keep people out so that not that everyone has access to the program. The applications are based on XML (extensible markup language), which is the universal language of Internet data exchange, and can be called across platforms and operating systems regardless of programming language. If you haven't heard a lot about XML, it's always growing. Microsoft is putting implicit XML support in the .NET, and it's just a way of setting up your data to make it easier to communicate.

Most programming languages are now supporting XML for data exchange. If anyone is familiar with the old life specifications, they moved from old life into XML life, and they have a specification available. The Accord Group, I believe, has XML specifications for life insurance uses.

This allows me to create a system or group of components, put them on the Web and then rather than relying on the assembly of components or objects, my programming can be based on the reuse of these services. If I develop a way to get my rates, I could put that on the Web service so that any application done by my company across the Internet can access my rates. I have to code it only once, make it available as a service, and then any other programming department or any department that wants to access those rates can do so through the Internet. The Web service can also grab those rates from my File Transfer Protocol (FTP) site, and they don't have to recode it themselves or find it themselves. When I need to change my rates, I can just post it to Web service, and anyone else accessing my Web service has the new rates. I don't have to redistribute these rates across my distribution network.

Another advantage is that I may want to get these rates out to brokers, who have their own programs to get my rates. Now they can call our Web server, access it as a service, and get my rates out. This can save in distribution costs. These services are invoked over the Internet by means of an industry standard protocol, such as SOAP, XML and UDDI. Again, none of these protocols were created by Microsoft, although Microsoft has come on board and is very big supporter of Web services.

SOAP is an XML-based messaging technology that specifies all the necessary rules for locating Web services, integrating them into applications and communicating between them. UDDI is a free public registry where one can publish and inquire about Web services. By using the UDDI specification, you can say, "There's a Web service out there called 'My Term Rates.' Where would I find it, and what do I have to pass in to get this information out?" Programmers can use that to communicate to your Web service.

I was talking of some examples. One example is a set of country or state/province and city tax tables that are provided to e-commerce shopping cart developers so they can accurately factor sales tax from online sales without being concerned about extensive data maintenance overhead. The government could, in theory, put a Web service on any e-commerce site that wants to get the latest city tax rules, so that the site could access this service instead of having to recode the information for every e-commerce application.

An insurance company can provide the term rate tables so that any broker can access the latest rates and give quotes from its own Web site. That way, you always have control over what rates your brokers are getting. You don't have to worry about old rates. Are they using an old disk? Are they using old rates? If they're always getting rates from your Web service, they will be using updated rates as long as your service is updated. This is true of any rates, including term, annuity, disability or critical illness.

These are just some examples of good ways to use Web services. I think that with Microsoft making that technology a lot easier, that's what is going to get out, that's what the company is pushing. In essence that's what the one degree of separation is. You have one degree of separation from your distribution to you. If your brokers are getting your rates from you, as soon as the rates change, your brokers have the new rates right away.

What are the benefits? You can allow your applications to share data and evoke capabilities from other machines, so if you have a Web service on one machine, and you have a different Web service that you need for your application on another machine, you're sharing the data among many machines throughout your company. If your company has a network of servers in different locations, you can split out the work across many locations and not have to worry about it because you can just grab that Web service from whatever server it resides on in your company.

Applications using the service are not concerned with how the service was built, what operating system or what platform it runs on, and what devices are used to access it. Because these are technologies across the Internet, if you have one UNIX machine that's providing a Web service, and you have another that's a Microsoft Internet Information Service, it doesn't matter because they're accessing the service through the Internet, through this common interface You don't have to worry about what it's running on. Microsoft is not the only technology that's allowing you to create Web services. Other languages let you create Web services, so you don't have to stick with Microsoft if you don't want to.

Although these services are independent, they can be coupled to perform a particular task. Like I said, if you separate a lot of tasks when you're building a system, you can take one calculation from one service, one from another and one from another, and then just put them together. It is possible for developers to choose between building all of their applications or consuming Web services from some parts or another.

Web services deliver dramatically more personal integrated experiences to users via the new breed of smart devices. I think, increasingly, devices will start talking to each other, whether it's your toaster talking to your oven, whether it's your PDA talking to my PDA. This can save time and money by cutting development time. You're not constantly redeveloping, and it can increase revenue streams by making your XML Web services available to others. Your company may charge to do certain services, calculate certain rates, prepare pension estimates, whatever it is—you can charge per use on a Web service because you can track who's coming in with a login code of some kind.

In conclusion, I think that .NET is poised to sweep the industry because it allows developers with different skills and backgrounds to work together. Microsoft is focusing much of its energy on this technology, and when Microsoft focuses on something, just through sheer money and effort, it usually comes out on top. If your company hasn't looked at .NET yet, you might want to start because it is gaining support all the time.

MR. FARRIS: Dr. Derrig is senior vice president of the Automobile Insurers Bureau of Massachusetts and vice president of research for the Insurance Fraud Bureau of Massachusetts. Prior to joining the bureaus, he taught graduate and undergraduate mathematics at Villanova University, Wheaton College of Massachusetts and Brown University for a total of 13 years. He earned a bachelor of science degree in mathematics from St. Peter's College and a master's degree and doctorate from Brown University.

Dr. Derrig co-authored several studies on subjects ranging from fraud to fuzzy logic. His research interests have involved the use of seat belts, traffic accidents and fraud detection risk premiums. He has lectured extensively on insurance topics to professional actuarial groups, trade associations and law enforcement personnel, and has held seminars at the University of Barcelona, the University of Hamburg, the University of Montreal, the University of Tel Aviv, and the University of Pennsylvania, the University of Illinois, the University of Texas and others in the United States.

He is a member of the Mathematical Association of America, the American Statistical Association and the Association of Certified Fraud Examiners. He serves on the Insurance Fraud and Auto Injury Study Committee of the Insurance Research Council.

DR. RICHARD A. DERRIG: Let me start with an advertisement. The *Journal of Risk and Insurance* is going to have a single issue, September 2002, devoted completely to insurance fraud, modeling and data mining. The authors of the papers will all be at a session of the Casualty Actuarial Society in November, plus a few more who have really developed the field over the past few years.

Let me start by just saying that I'm not going to be very extensive like my two colleagues here. I'm going to be simple because the problem can be described simply, and I'd like to give you the same problem twice. The problem is the detection problem. How do you detect something?

The first detection problem that you've all been involved in is passing exams. You took exams, you answered questions, the questions were evaluated, and the examiners sent you a note saying you passed or you failed. This could have been a lot easier if they just gave you a check off box in the beginning where you said, "I will pass or fail." That variable, pass and fail, is what's called a latent variable: that is, no one is going to see it while you're trying to detect it, and somebody has to make a decision at the end of the game to decide whether you've passed or failed.

The same kind of problem is evident in fraud detection in claims, as I'll talk about. Nobody files a claim checking off a box that says, "This claim is fraudulent" or "This claim is not fraudulent." Just like passing and failing, it's a latent variable that you're trying to figure out given the information that you have at hand. Two developments that I'll talk about are papers that are in this special issue and that I think might have general interest.

First of all, I need to tell you what fraud is, and for me—and this is not universally accepted. Fraud is criminal and has four principles that have to be proven in court for you to convict anybody:

1. It has to be a clear and willful act.
2. It has to be against the law.
3. Money or value was obtained.
4. The gains were made under false pretenses.

Anything less then all four of them should go under the terminology of abuse. Fraud and abuse are things that we do not want in the insurance system even though they're there, and so we'd like to detect both of them.

Among the types of fraud that are out there is insurer fraud. Martin Frankel's company, which you may have read about lately, is a prototypical type of a fraudulent company in management. We're not talking about that, although we can.

It's hard to believe, but there really is some agent fraud out there where they keep the money or they really don't write the policy. There's insider fraud, company fraud, embezzlement or inside/outside arrangements.

The one we work on is claim fraud, where the claimant, the insured or providers, or rings of organized crime get together and try to extract money from insurance companies under false pretenses.

How much is there? We could spend days on this, but all I want to tell you is that it's not 10 percent, which was a number made up in 1976 by the General Accounting Office, passed from person to person, and still appears in almost every trade press article. What companies do about fraud is they investigate, negotiate and litigate. They investigate, meaning they acquire information from the claimant or anyone else associated with it. We've done studies on auto bodily injury liability claims that show investigation reduces payments by about 18 percent.

They negotiate if they have a negotiable position, like they do in liability insurance. Negotiation reduces built-up claims, which means excessive treatment or excessive exaggerated injuries, by 22 percent, at least in our study. Finally, companies have the ability to litigate, take people to court and claim that they're filing false claims.

The techniques they use and the object of detecting a claim that may be fraudulent are referrals to special investigative units, which grew up beginning in Massachusetts in the 1980s and are now almost universal. Independent medical exams, medical audits, peer reviews, expert systems, surveillance, recorded statements, sworn statements and accident reconstruction all are used.

What they have in common is that companies have to spend money to do them. The decision that needs to be made as information comes in is whether you want to spend money to acquire more information or whether you don't.

The top 10 defenses are the adjusters, and for us today, computer technology—associated with criminal investigators—data information, experts, and if we're really talking about criminal behavior, judges, lawyers, and legislators, prosecutors and special investigators—all of those people combine to use the output of a fraud detection system.

I also have the top 10 ideas to give you. The first problem is that fraud is ambiguous and ill-defined, and so, when people actually talk about it, even analysts, they usually are talking at cross-purposes. It should really be reserved for criminal behavior. The ambiguity inappropriately allows players (players meaning the same people I just listed, including companies) to seek solution in law enforcement, when, in fact, it's the civil system that's the problem.

Criminal fraud is several orders of magnitude less than estimates. If you've read anything about fraud in the trade press, you undoubtedly heard billions and billions. It sounds like, you know, Carl Sagan and his stars, but it's not true. It's very small relative to those kinds of estimates, but the magnitude of unwanted and inappropriate claims is that large. Fraud and systematic abuse can be and should be mitigated by computer-assisted adjusters and special investigators. This is where the technology needs to come in. The fraud detection problem is a problem of classification and not claim identification.

Even though I tried to sell you that in the beginning, it's really the wrong problem. Now, you're not trying to detect a particular claim, you really need to be classifying all the claims. And smart classification allows smart allocation of loss adjustment expense or buying information.

Three new research papers have produced the next three results. The first one is the first crack at something that's very difficult. That is, how do you think about, measure and talk about fraud deterrence as opposed to detection? Deterrence is when people don't do what you don't want them to do. One paper in the group is talking about that, which I won't go into detail, but you can check audit ratios as to what's being detected. When they're constant across risk classes, this is the optimal deterrence that can be had.

I will talk about the other two papers. A study has been done out of Belgium that shows generally that a cla ims scoring models—that is, if you assign a number for each of the claims based on features of those models—can be generated almost as accurately by logistic regression alone as opposed to lots of fancy techniques. The one I'd like to talk more about is one that's a statistical method that can optimally classify by fraud indicators present and detect also changes in incoming data.

Those two studies represent two different approaches to all of this. One approach is that you have features—that is, characteristics—and that's all you have, so you organize classifications by organizing the claims via their characteristics. The classic example of that is clustering. The one I favor, of course, is fuzzy clustering. But in any event, it's just the data, the features and your cluster.

The other way to do this is that you have the same data, but you also have some evaluation, some method of deciding whatever it is you're looking for about each of those claims. When it's expert judgment for examples of fraud/no fraud, you'd have that attached to all the claims, and you can build a model. If you're into artificial intelligence, we're talking about supervised or unsupervised modeling. The nice thing about this is it says that they're connected in the following way, that every set of feature data that satisfies only one condition comes with a scoring model that optimally characterizes those data. It's that theorem that was proven that allows us to be able to evaluate the data versus any supervised model, and the data changing in themselves. That's why this all works, and it's why at least I'm excited.

The value of these kinds of systems is to detect fraud early and do something about it, but it's also available to audit and to measure what's happening within the company, sorting to efficiently refer to special investigators, and as an enterprise out there, these kinds of things can provide evidence to support denials of claims and to protect against bad-faith suits.

The way that this has been done in the past, and the past is the present pretty much, is experience in judgment, but the newer way that we'd like to see used is artificial intelligence systems that employ regression models, fuzzy, neural networks, expert systems, genetic algorithms, and all of the above.

There is a survey of companies just done by the Insurance Research Council about fraud. And if I remember correctly, the number of executives, CEOs, et al., who were ranking systems development ranked mathematical and algorithmic-based systems last. Four percent of all these people thought that they would be helpful, so there's a long way to go here.

Where is it used? A claim comes in. The first thing that can happen to it is nothing. We call those duds. In fact, for personal protection in automobile insurance, about 25 percent of them don't go anywhere. There is something that I don't need to go into here, but there needs to be a triage in the beginning for trivial claims versus claims where the question of whether you acquire information is worthwhile. There should be some kind of pre-data mining that separates those two. In our situation the express claims are about 20 percent of the claims.

After that, if you've got target claims to look at, you can use a data mining algorithm to start sorting them into groups. Just like you can sort all the students into groups, in this case it's two groups. Routine adjusting is the output of it. And if it's simple, the data mining comports with the information that you have. Then you pay them. Although this is usually ignored, this is where all the bucks are because the bucks are in saving money acquiring all that information. That list I gave you, those kinds of things; and in your business, too, acquiring information is costly, and when you cannot spend it, that's as valuable as deciding when to actually go out and spend it.

If there is an investigation, that can happen. After investigation, if everything is okay, you pay it. But things get complicated if you acquire information that shows that there is either abuse or fraud. All the way at the bottom is guilty, guilty as charged after prosecution. So that's the system, and what I'm talking about is what goes into that data mining because that's when you want to classify the incoming objects or the claims into different bins, and then you have different strategies to deal with each one.

Hard fraud, by the way, is what's called guilty. Suspected hard fraud is referring for criminal, and suspected fraud and abuse is the whole bottom part, which includes negotiation as the output of value.

As an example of the simple sort, for auto personal injury protection claims, you can sort and pay them if the expected medical is less then a thousand dollars, there is no disability and, most important, there is no attorney. For the other part, the harder part, the data mining part, essentially no matter what you do, at least in my mind, it comes down to a simple scheme of taking databases and summarizing them in the same kind of statistical method that you use to summarize distributions into expected values and other moments and then have an output that's graded that decisions can be made on. Out of that comes nonsuspicious claims and routine claims, or suspicious claims and complicated claims.

The principal one that I wanted to talk about is fraud classification using the principal component analysis of RIDITs. The idea for this actually came from educational scoring, so the example I gave you is not far from where these ideas came from. The idea is, suppose you were trying to decide whether or not Mark gets to be a Fellow and pass some more exams. What questions do we ask him? Based on history, perhaps we can decide that is or is not a good question to ask to discriminate between those who pass and fail. You know half the people pass, half the people fail, and the question doesn't really tell me anything. That problem back in educational testing, if you think about it, is the same problem of trying to detect fraud using particular features of the claim.

An example in our business is if no police respond to the accident; if they don't respond, there's more likelihood that you can generate a false claim than if they do and write an independent view of it. That sort of thing allows features to be built. In our case, and in the example that's in this paper, there are 65 different features of a claim that can be used.

The overview of it, just to get you to the bottom line quickly, is suppose you have N claims and T features. In our case, 65 features of it, and the responses of each feature are K_T, so I can have two, three, four and five, and I mentioned I have to have an assumption. The only assumption that's needed is that the responses are monotone in whatever the latent variable is.

Let's say I have five responses, and I'm looking for fraud versus no fraud. We're pass/fail. The first response has to be the most likely fraudulent. The next would have to be the next likely, and so on. It doesn't matter which order it is, but it has to be monotone. And if they're all monotone, the theorem applies. That is, you get a convergence given the scoring systems, and you get a unique scoring that bins all of the data according to the feature, according to the underlying latent variable. There is a way that you first of all transform the scores, which are monotone, into what are called RIDIT scores, which are nonlinear.

The key part of this and the output of the theorem is that the success of iterations trying to figure out which ones are more important, which of the features are more important to distinguish, converges, and although everyone knew that part, what was proven for the paper was that it converges to weights that are the principal component of the array of cells with the RIDIT scores in them. It's amazing because, instead of this complicated iterative procedure, you invoke the theorem, crank up the computer, get the principal component of this thing, and there are your weights: you've got the scores and it's all over. The scoring is the weights times the RIDIT scores, and what it does is it gives you a score where the partition is zero, and above and below zero it gives you the two partitions of the latent variable, fraud, no fraud, pass/fail.

As an added attraction, the size or the distance away from zero is also informative, informative in the way of determining whether this is a really good distinguishing feature or not. The farther away from zero those scores get, the better the feature is. So there's a lot of information that's all packed into the output of this theorem.

Let me show you the other one. This is actually the way it's done (Table 1). When looking at a large number of visits to a chiropractor, for example, the RIDITs score is - 0.56 in our data. Just as an example, treatment 1—lots of visits to a chiropractor—didn't really do anything differently as weights using regression with a supervised output target versus the PRIDITs (principal component analysis RIDIT). Neither did the second treatment variable. But as you work your way down the data—if you think of the data on the left and the model on the right—what this is telling you, besides the fact that you can generate a model from the left side only without the right, is that your right-hand variable, or your right-hand model, is not tuned to the data you're getting because you're getting a different answer from the internal scoring model that's representing the structure of the data from the one you fit, however you fit it. In fact, in this case, these numbers are coming from fitting it on prior data. It gives you a nice way of testing whether or not the models you're using fit the data because the data have structure and have an implicit, impounded model, and you can always compare that one to whatever you build.

Table 1

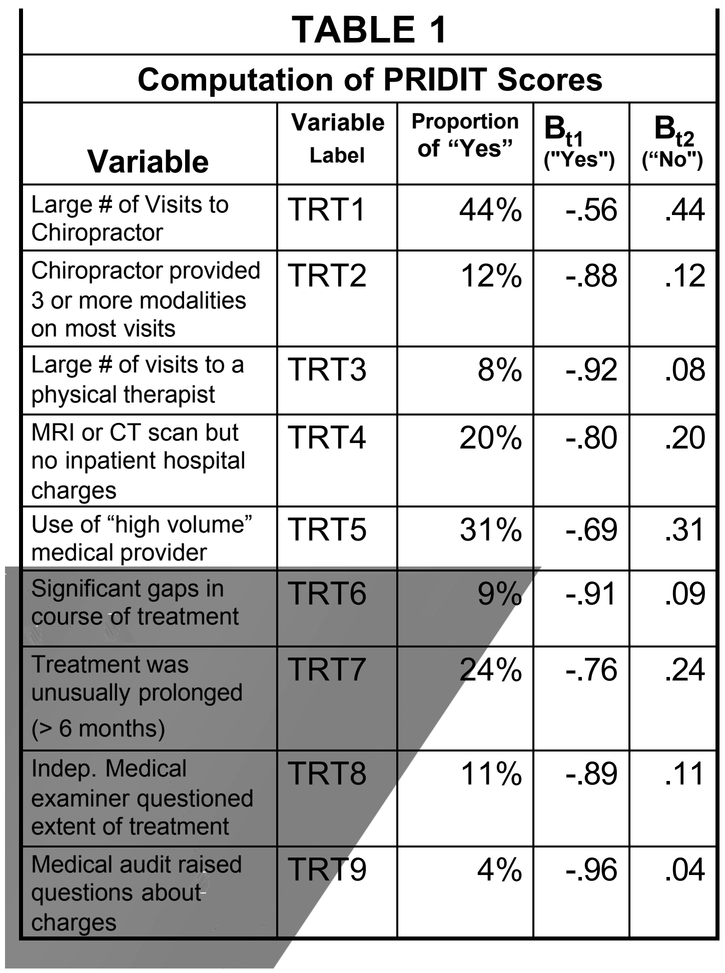

TABLE 1

Computation of PRIDIT Scores

Variable	Variable Label	Proportion of "Yes"	B_{t1} ("Yes")	B_{t2} ("No")
Large # of Visits to Chiropractor	TRT1	44%	-.56	.44
Chiropractor provided 3 or more modalities on most visits	TRT2	12%	-.88	.12
Large # of visits to a physical therapist	TRT3	8%	-.92	.08
MRI or CT scan but no inpatient hospital charges	TRT4	20%	-.80	.20
Use of "high volume" medical provider	TRT5	31%	-.69	.31
Significant gaps in course of treatment	TRT6	9%	-.91	.09
Treatment was unusually prolonged (> 6 months)	TRT7	24%	-.76	.24
Indep. Medical examiner questioned extent of treatment	TRT8	11%	-.89	.11
Medical audit raised questions about charges	TRT9	4%	-.96	.04

The last one, treatment nine, shows that we actually went the wrong way. In other words, we thought something predicted fraud, and, in fact, it's the opposite. This is just to show an array that asks, "What's most important according to the data?" Same accidents, same claimant, and mostly injuries, and it's not quite the same as what experts would have predicted.

Checking whether or not the data are changing can be as simple as looking at the old pi^2 2x2 table. If I classify it by fraud and nonfraud according to the PRIDIT model versus some other model, if they both identify things, that tells you something about how your model is lined up with the data and whether or not the model you built is reasonable. You can actually calculate confidence intervals on this stuff. Likewise, if you have another model come along, you can calculate the same alpha, which is just a cross-product, and decide whether or not you're matching the data better, higher alphas, or you're not matching the data as well.

Finally, the other paper that I wanted to at least mention is a paper that compared what's called state-of-the-art classification techniques on data. This is the other kind of detection model, where there's a preprocessing of the data. You already know an answer on some training data, and you're trying to apply it to a general set of data. The objective here was a comparison of binary classification algorithms. The finding, to sum it up, is that there are small differences in predictive performance; simple techniques score very well, meaning logistic regression; and decision trees—which was a shock to these guys in Belgium—really didn't work very well because they're touted as doing very well. In any event, that's a little list of things that were used to test. The nearest neighbors are Bayesian learning, and the conclusion was that logistic regression is just as good.

I just want to close by pointing out that the expert preprocessing and performance-boosting techniques may improve otherwise, give other results. This study did not allow the modeler to interfere and to boost performance, as it's called. That is, it was a pure test of the algorithm versus one algorithm versus another algorithm. It's certainly the case that if you've got an expert that knows how to preprocess the data, which means change the variables to ones that are more appropriate, or the performance-boosting techniques, which are complicated, with a person you can probably outdo logistic regression. But as an algorithm, it isn't better. The reference that I would recommend if anyone has an interest is a new book by some statisticians at Stanford called *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. It's probably one of the better mathematically oriented books written in the past 10 years.