RECORD, Volume 30, No. 2*

Spring Meeting, San Antonio, TX
June 14–15, 2004

Session #3PD
Actuaries versus IT—The Perpetual Dilemma

Track: Computer Science, Futurism

Moderator: Michael N. Hartfield

Panelists: Emil Burns Kraft
Susan M. Lee
Frank G. Reynolds

Summary: Actuaries and IT people rarely see eye-to-eye. Neither understands the other's perspective, with frustrating results for both sides. Yet, with some tolerance and understanding, both can profit from the other's approach.

MR. MICHAEL N. HARTFIELD: Today's first speaker, Emil Kraft, from Milliman, was a consulting actuary, but he's made the transition over to software and managing software development. He holds an A.S.A., Member of the American Academy of Actuaries (M.A.A.A.), and M.C.S.D. certifications and had some success back in 1995 as the highest-ranking score in the International Course I competition. So he knows what actuaries go through.

MR. EMIL KRAFT: I joined Milliman USA, now Milliman, in about 1996, starting as an actuary. In late 1998 I transitioned to a technology group, and for the balance of the last six years I've been managing software analysis development and integration practice. Just recently, I started with an actuarial team working on a couple of new product initiatives, but the balance of my work over the last eight years has been working with both actuarial and IT folks. I've been wearing both hats to solve actuarial problems from a systems perspective.

I love the title of this session. My message is that there's hope, that we've had a lot of good experiences. So my presentation theme is simply going to be that many of the conflicts between actuaries and IT staff are simply due to the nature of the structure of systems, and that if you change the structure to be consistent with the nature of a lot of the business problems, much of the adversarial quality of the relationship between IT and actuaries can be reduced or even eliminated in a lot of cases. I think other presenters are going to raise a whole litany of fantastic examples that I'm sure everybody cancommiserate with, but in this presentation I'd

like to celebrate some of the successes over the last few years.

My background is pretty heavily in rating and underwriting systems. I'm more on the health side, and rating and underwriting systems are a pretty serious systems issue. Systems are becoming a lot more automated, and there's a lot of benefit to that. So that's where a lot of my work has been, and that's where I've seen a lot of successes in actuaries and IT folks working together. Specifically, the theme of my presentation: I'm going to put it within the context of rating and underwriting systems for a number of reasons. Rating and underwriting systems are historically very painful due to their analytic nature. It's pretty easy for a businessperson to spec-out sort of a call center support application, and it's pretty easy for programmers to bang it out. But there's a very highly analytic nature to rating and underwriting systems, especially for large groups.

So you get into a situation where the actuarial people have a 90 percent understanding of what they want but have a very difficult time communicating it, and the IT folks have a 0 percent understanding of what they want and a limited ability to understand the business side. And that gives rise to a lot of problems. Also, specific to rating and underwriting systems, usually sales is involved. It becomes a three-way confrontation where actuarial and underwriting are pitted against sales, pitted against IT, etc., and commonly the two that are closest will band together against the third. It's a terribly political situation a lot of times. So it's a very contentious topic even outside of the actuarial and the IT realms.

These are serious, serious systems. Billions of dollars of revenue can be driven by rating processes, even for a midsized health plan. So we're talking about the drive train of their gathering of revenue. And, last, it's illustrative, because a number of technologies have been introduced in recent years to help address the nature of these systems and help align them more closely with the business problems at hand. So I think it's timely for an example of what's successful in relations between actuaries and IT folks.

I want to start out by getting a little bit of perspective on some common rating system alternatives that are pretty much the types you're going to see in the marketplace today. One is Excel- or Word-based processes, where people are going and getting a data extract and pasting it into Excel, and calculating and changing some numbers, and then taking numbers and putting them in Word, whether it's in an automated fashion or more manual fashion, and then that Word document goes off to sales and whatnot. That's more of an Excel–Word–based process. And then the second is production systems, which is either client server or Web-based. It's going to be compiled code, and it's going to automate a lot of those processes.

These enjoy various advantages. Automation: that's essentially, am I doing something, or is the computer doing something? For sure, production systems are usually fully automated, but there are a lot of manual processes involved outside of the realm of production systems. So at best they're only partially automated.

Another advantage is transparency. Transparency essentially means as a business user, as an actuary, can I quickly see the structure of the calculations or the structure of the business logic in general? And for a specific group or case or application of that logic, can I see the numbers and how the numbers flow through to come to my final conclusion?

This is an area where Excel dominates. Excel is completely transparent. That's why it's one of our favorite mediums as actuaries. Production systems are typically anything but transparent. They're usually totally opaque, and unless you understand the C code or whatever's underlying it, you don't have a lot of transparency in your production system, which gives rise to a lot of frustration. It's very easy to develop an Excel: it's a very flexible medium for modeling financial calculations, but it's not easy to develop for a production system.

Confidence in the accuracy: it's pretty easy for an actuary to have a lot of confidence in the accuracy of an Excel workbook because it's transparent. They can get in with the auditing function, they can look at it, they can drill down, and they can have a comfort level that the calculations are correct. It's a lot more of a time-consuming process to gain a level of confidence with a production system. Testing cycles can be months long.

I've gone in and found unbelievable errors in some systems. We did a rating system, and I found for a small life insurance company that their production system was issuing rates to underground miners and crop dusters at the same rate as schoolteachers because their systems weren't programmed to exclude underground miners and crop dusters. They'd been using it for years, and I'm sure they weren't losing their shirts because they were selling insurance to crop dusters. But there are problems in production systems that are hard to find.

The next advantage is flexibility. It's very easy to add a product or change a calculation in an Excel workbook, while a typical lead time to get IT to change something in a production system is a couple months, sometimes longer. Another one, language independence: that's a little technical, but Excel workbooks survive as technology platforms change. The same Excel workbooks or Lotus 1-2-3 workbooks that were developed in 1990, from which time you might have a production system that's more in COBOL, can be upgraded to work in the Visual Basic era, and now can be upgraded easily to work with a dot-NET and a J2EE. So one of the benefits of Excel is a degree of language independence that production systems don't have.

And then, last, you get to the benefits of production systems: scalability, reporting, having centralized security, versioned updating, where you're able to make mass releases or updates at a single point to various folks, and, if it's Web-based, having a single installation. So those are some benefits of production systems.

What you find is that a lot of times people will split right down the middle. It

probably makes intuitive sense that actuaries really like transparency, to be able to easily control it and be able to add things and have a lot of flexibility and confidence. IT folks really want to have a lot of production system infrastructure, because that's what they're most comfortable with from their perspective.

This is somewhat of the split that gives rise to a lot of the confrontation. It's very difficult to have all the benefits of these. And so the gist of the remaining presentation is in a number of areas where actuaries really want to have a lot of control over their technologies now and approaches that allow you to develop systems that bring a lot of these benefits together and can please both sides.

Now, in talking about what actuaries want to have, there are really only a handful of areas of business controls specific to rating and underwriting systems where actuaries and underwriters really want to have a lot of control. The first of those four areas is rating formula calculation engines. They want to be able to control the calculations that are rating the groups. Another area is the analytic environment. If you've got, for example, a large group where there's going to be a lot of actuaries, say, a health care group, who are going to want to look at running all the groups in a book of business through the system, looking at the macroscopic picture, making some high-level adjustments, maybe potentially smoothing for rate variations over time—that's an analytic environment to support that activity.

Similarly, on the underwriting side, if you have an underwriter going in and adjusting the threshold, after which there's pulling, or we're excluding certain people either because they're dead or from the financial mix or for some other reason, that's an analytic environment. That is really actuaries' and underwriters' domain. So it should ideally be something they control.

Another one is dynamically generated documents. That can be anything: sales illustrations, communicating information to the client, documentation, etc. It's very handy. Underwriters, salespeople and actuaries have a lot more of an understanding of what that should look like than IT folks do. And, last, parameter tables: updating rate tables and the like.

The classic problem with rating engines is this: you have a business user who writes down the rating logic, say, in a Word document, and they're human, so they make mistakes. Then that Word document goes to a programmer, and the programmer writes a program that's supposed to do the logic, and they make mistakes. Then the business user looks at what the outcome is of the computer code, and they say, "Well, this isn't right." And then that begins a seemingly endless cycle of the business user trying to figure out what's going on in the programmer's head and the programmer figuring out what the business user meant, and it forces both potentially very talented professionals to have the discomfort of working in environments that they're just not trained for and they're not familiar with. And so this is an area that gives rise to a lot of conflict between actuaries and IT folks.

One solution is that different technologies now allow you to actually compile Excel workbooks. Once you have an Excel workbook, you can automatically migrate that logic into production system code without manual intervention. In this scenario, a business user will develop an Excel workbook with the logic. They'll pass it around for peer review. People will make any adjustments or tweaks needed. When all the actuaries have signed off that this is the logic they want to implement, you can employ an automated process using different software tools to dynamically generate production system code that can be used in production systems that is 100 percent accurate. One of them is KDCalc. It's a commonly available software tool from Knowledge Dynamics, and it serves J2EE and dot-NET. Before we were aware of KDCalc, my team actually developed an Excel compiler called X Logic that worked for both Visual Basic and dot-NET. There are some subtle differences between the two, but with an Excel compiler there are a lot of benefits.

For example, we had a client, Central Benefits, for whom we did a rating system. They had rating logic that started in 1990 and a Lotus 1-2-3 workbook. Twelve years and many owners later, it had evolved into a mess. A six-megabyte Excel workbook was a complete disaster. It did all the right calculations, but it wasn't pretty. It had been through about 10 people's hands. The client wanted to migrate its mainframe system to the Internet. —The Excel workbook was built to test the mainframe system because it needed a degree of transparency not provided by the mainframe system—. The client changed and tested the Excel workbook so it was up-to-date. We used an Excel compiler to dynamically generate 60,000 lines of error-free code from the workbook in an hour and a half while we were at lunch. Typically that would take months for a human being to code manually, and they'd make all kinds of mistakes. So it was unbelievable. We integrated the engine into a Web-based rating and quoting portal.

The maintenance continues today in Excel. Whenever they want to make a change to the rating engine, they make that change in Excel, and then we can automatically migrate the changes through to their production system. Excel became, and is now, the design, development, testing and maintenance environment for the production system calculation algorithms. And that was really amazing. We did a few rating systems where we coded the rating engine manually, and that gave rise to us trying to find a better way, which is why we wrote our Excel compiler. Now there are other Excel compilers on the market that serve other technology platforms, and it's unbelievable. If you've ever coded a rating engine to see something go from Excel right into production in hours or minutes or seconds, it's a beautiful thing. That's one thing that really helps eliminate a lot of the contention between actuaries and IT folks, which is completely unnecessary.

FROM THE FLOOR: I'm just trying to visualize compiling an Excel workbook into some other language—what that process looks like, what format it needs to be in. Do you just throw the old Excel workbook at it, and it pops out, and it's clean and done? I'm just trying to get an idea.

MR. KRAFT: Well, here's the answer. Pretty much all the Excel compilers are driven solely off worksheet functions. I'll speak about ours just because that's the one I'm most familiar with, but all of them are generally alike. You have an Excel workbook, and in ours you say, "Here are the output ranges that we want to harvest. Or here are the output ranges we want to put in a downstream system or put in a dynamically generated document or create a report off of." And then the compiler will iteratively work up through the precedents to find all the inputs. Once you have all the inputs and outputs, you have your interface. And then with the inputs and the outputs and all the calculations in the middle, the compiler will dynamically generate dot-NET or Visual Basic; if you're doing a legacy system or JAVA, it will create in dot-NET a JAVA Web service or the engine that can be snapped onto a Web service.

So the idea is that you would send a Web service or a programmatic interface, say, an XML document with all of my input cells that were identified during the compilation process, do the calculations, then it will return to you an output XML document that has all your outputs. So it literally takes your Excel worksheets, identifies inputs and outputs, and gives you the ability to call those without having to call an Excel workbook.

FROM THE FLOOR: My question is about a situation where you've got persistent data existing on databases, files, whatever. Is it as easy to plug in, for example, the calculation engine as in the environment where you've got a nice, neat XML format to easily map to the calculation engine that you compiled and translated over to some other code?

MR. KRAFT: Great question. This compiling of Excel spreadsheets closes one scope of system development very, very well, but it's a limited scope. You're going to have a database that's going to have quote information if we continue the analogy of rating systems. It's not going to be a replacement for that. The difference is that potentially, instead of a human being having a macro in Excel that will pull the right quote information into the Excel workbook for them to do a bunch of manual work and then dump up some exhibits, just as that's an integration with a database or a data repository, you would integrate the database with the Web service.

One case where it's ideal is when you've got sales people, and they're in the field, and maybe you get a thousand hits a day on small group business quotes. That's not something where you want an Excel workbook as your calc engine. So that's an example where there are a lot of benefits to taking that calc engine, compiling it into Web service and then having a more stable production calculation engine doing those calculations. And that's all that it's a replacement for: you'd still have the database. You still have to design a Web site. There's a lot of surrounding infrastructure. But what can be tens of thousands of lines of code you get for free, and that's been a tremendous benefit.

The problem with analytic environments in production systems is they're typically

very limiting, very opaque and inflexible. The classic example is for analytic interfaces: people are used to using Excel, since it's very flexible, very transparent. The worst thing you can give an underwriter who's used to working in Excel, who loves Excel, is a Web-based analytic interface. They will hate it, because you don't have the freedom you have in Excel. Every time you change anything, the page has to refresh, and it slows down people's productivity sometimes; it just drives them totally crazy.

What we found in terms of a good solution for this is to have production systems dynamically generate Excel-based interfaces. Just as you can have a system dynamically generate a Web-based interface, you can have a system dynamically generate an Excel-based interface. This isn't a cure-all. If it's a pretty simple analytic environment, then a Web page is probably fine. But if it's an analytic environment with any degree of sophistication, our clients—we've heard that other folks are doing this as well internally—love having these Excel-based interfaces. So this is where the Excel-based interfaces are dynamically generated by the production system. They're downloaded by business users—potentially a right click like on a Web page and downloading it to your local machine. You can then complete all your analysis in the Excel workbook, then upload that back into the system. The system then harvests all your changes and your comments, and the content is extracted and stored in databases and flushed onto downstream systems and the like.

This has worked very well in production. For example, at Blue Cross/Blue Shield of Rhode Island, we achieved a consensus on a highly generalized data feed to support the analysis. This is where you make the IT folks and the actuaries happy. If the actuaries can say, "All we want for the IT folks is for you to give us a highly generalized data dump, in Excel," then we can build the macros and the code that will build our production system environment with macros in Excel. That's a very nice split. So, in Rhode Island, we developed a consensus on a highly generalized data feed to support their analyses, and we brought in interface standards so the IT folks had buckets to fill, and the actuaries and underwriters knew what buckets they'd get. Then the actuaries built macro libraries to help the actuaries and underwriters retrieve and manipulate the data along the lines of expected use.

IT programmed the production system to generate Excel workbooks with a specified data feed and then kick off a macro that was written by underwriting and actuarial that programmed virtually all the analysis worksheets against the specified data feed using macros. The upshot is that the actuary fills a portion of the workbook with data every time, not having to know much about the business use for it, and the actuaries and the underwriters go ahead and write a macro that fills their analysis environment with all the data exhibits they'd like to see. So now actuaries and underwriters perform their analyses in Excel.

FROM THE FLOOR: Does the applicability of this downloading of an Excel spreadsheet apply additionally to Visual Basic macros within an Excel file? Lots of

times a person will write an Excel file and use a macro and write it in Visual Basic. Does this process where you actually can upload into a production mechanism, an Excel spreadsheet also apply to that kind of macro?

MR. KRAFT: Well, if you're going to upload a production workbook, you can get at anything. Could you give me an example of how you might want to use this?

FROM THE FLOOR: Let's say you're generating a routine—take a simple example—to calculate reserve credits for YRT reinsurance on GAAP. An Excel macro can get very complicated because you've got an awful lot of multiples to a base table and so forth. So you've done most of that outside of the basic spreadsheet; you've done it by means of a Visual Basic macro instead. Can you upload all of that into the production system you're talking about?

MR. KRAFT: There are a couple different questions there. Maybe you've only asked one of the two, but one question would be I'm doing my analysis, I've got a production system that is expecting these reinsurance rates. So you have an Excel workbook, and you're doing all your analysis, and then you want to commit these Excel rates after you finish running them. You've run your macros, you have your rates, you're looking at them. Now you want to load them in a production system—the rates, not the logic. You can upload the Excel workbook, and any production system that's configured properly can harvest those rates, harvest any of your comments from the cells, and harvest any other changes you made to be used downstream. Is that what you're interested in uploading, the rates or the logic?

FROM THE FLOOR: Well, if the tables are actually in the Visual Basic logic as opposed to being in the Excel spreadsheet, can it pick that up? That's really my question.

MR. KRAFT: It sounds like what you're talking about is you're wanting to migrate the logic in the macro into a production system. That's a more intractable problem. What we have typically recommended and what we've seen is that the vast majority of the complexity when you talk about Excel compilers—and this is a classical question—is, Does it also compile macros? That sounds like what you're getting at: can you also migrate the macros into production? What we found is that migrating macros into production is much more difficult than migrating the formulas into production, and that itself is very difficult to do with all the dependencies and whatnot. It's a very challenging technical problem to overcome.

So also compiling the macros is another whole order of magnitude of complexity that, to my knowledge, has simply not been done. What we found as a work-around is that the majority of the complexity in the workbooks I've seen can be put in formulas. And the macro language typically—and I say "typically" because I don't know your situation—can be boiled down to doing things like looping, but you can't do looping in an Excel workbook. So it will come down to maybe people using macros because they have a complex engine in the workbook, and they want to run

it 10,000 times for a bunch of different scenarios and then maybe some postoperative analysis. It's very easy to do the looping manually in a production system and just compile the heart of the calculation logic.

Typically when people want to compile macros, we've boiled it down to first compiling the workbook, which is 80 to 90 percent of it. Then the handful of looping statements or other types of purely programmatic control statements can be done with manual coding or copying and pasting the Visual Basic macro. This means that a lot of your Visual Basic stuff, which is now legacy technology, strictly Visual Basic, can be copied right out of the macro areas. So you can't compile macro code in my experience, but you can come pretty close, and there are some helpful workarounds. You get a lot of power with the workbooks. You get "If" statements in the workbooks. A lot of macro stuff can be put in the worksheet.

The upshot of the analytic environment is having a handoff between IT and actuarial that lets IT do what they do best, harvesting data from unwieldy databases and having it available in a production system, and letting the actuaries and underwriters do what they do best, which is manipulating those data in an environment that they're comfortable with. I'm going to skip over dynamically generated documents, because that's probably the least flashy and the least interesting from an actuarial perspective. Basically what we've done in terms of dynamically generated documents is just as you give people an Excel environment in which to do their calculations, the natural environment for editing a dynamically generated document is Microsoft Word.

So we've developed a document generation engine that allows people to configure dynamically generated documents in Word, just like Excel is the ideal development environment for calculations. Once you give salespeople and underwriting the ability to change and alter their dynamically generated document formats, that opens up a lot of potential in increasing the communication between sales and underwriting and stuff that just gets bogged down by IT, which is not capable of understanding the business need of communications of rates and stuff to the client or to sales.

In conclusion, by going with technologies and approaches that elegantly fit the business situation, it's possible to take a lot of the burden and a lot of the confrontation out of the relationship between IT and actuarial. Specifically in rating, if you give actuaries the ability to develop and test their production rating engines in Excel, if you give actuaries the ability to develop and test their analytic environments in Excel for a production system, and if you give actuaries, underwriters and sales the ability to develop, test, implement or change the production system, dynamically generated proposals, quote, sales illustration documents in Word and Excel, and you give them the ability to change their parameters in Excel, actuaries and IT people can work very happily together on rating systems. It eliminates not only potentially months from the development and implementation cycle, but it also helps both teams do what they do best and work

very productively together.

MR. HARTFIELD: Our next speaker is Susie Lee, an F.S.A. from Allstate Financial. For the last 12 years, she's been responsible for making enhancements to proprietary and vended software programs and tools. Her presentation will be from a slightly different perspective than Emil's and at a slightly different level of abstraction.

MS. SUSAN M. LEE: Emil's presentation focused more on the actuarial side. Although I am an actuary, I'm actually speaking more representing IT.

I'm going to borrow one of the points in Emil's presentation: he started off with his theme being that many conflicts between actuaries and IT exist due to the nature of the structure of systems. I'd prefer to change that to due to the lack of understanding of the structure of systems. I also believe that if you align your system structure to be consistent with the nature of the business—I don't like to call it a problem—you will minimize much of the adversarial quality of the relationships. I say "minimize" because I actually enjoy a little bit of conflict; I like the difference of opinions. It sometimes drives me in a different direction than I might have considered originally. I don't always think of these differences as problems. I view them as challenges or issues, and I'm actually more excited about challenges. Problems are not in the business world, so to speak.

So there are two key phases to the software development process. The first one is design, and the second one is construction. I'm going to review a couple of these with you so that you will have a better understanding of the nature and the structure of systems development. The design phase is often the most difficult part to predict and requires extremely creative people. The construction phase is easier to predict. You can employ less skilled individuals, so you should think automation there. Emil has given you a very good example of how they were able to automate certain construction elements in his Excel examples. We're going to explore these design and construction elements that we can then modify to help bridge our gap between IT's and actuaries' expectations of each other and what the results are.

For actuaries, our expectations of IT are that we should be able to adjust priorities to quickly implement changes, and that's an element of design. We want the programmers to fix bugs and technical difficulties. That's an element of construction. We think they should design programs that easily incorporate modifications. Again, that's also design. And then we want them to deliver error-free systems, which is a component of construction.

For the IT person, if I'm the programmer trying to cut some code for you, here are my expectations of the actuaries. I want you to write specs that include all reasonable, foreseeable modifications. That's an element of design. I want you to write me specs that I can program from. That's also design. And I want you to test all the system calculations. If there are 20 "if" statements, I'd better see 40

function points on your test case. So that's construction.

So what are some design fundamentals? First, in the design phase of your software construction process, you need to accept that the design will change. Your business doesn't stand still. You don't stand still. Why should the program that is built stand still? If you accept that the design will change, then you should be able to write specs that will include all reasonable, foreseeable modifications. You need to focus during the design phase on what is predictable and then adopt processes to control the unpredictable aspects. You should shift the emphasis of the deliverable. Instead of looking at whether you got more value from the end result than you put in, look at whether they delivered something on time and on budget. Coming from the IT department, I see this is something that happens frequently. I tell them, "I gave them exactly what they wanted, but it doesn't do what they want because they asked for the wrong thing." Or did I give them something better than what they asked for? Did my involvement in the project give them something of greater value?

To me that is a better measure than asking, "Did I get it to you by a specific date and for a specific dollar amount?" If it took me an extra day, but you got a lot of man-hours out of that one particular day, and you can do a lot more when you have the product in your hands, I think that's a better measure of success than sticking with those hard-and-fast rules. I would also encourage you to do this when you're negotiating for consultants for a variety of software expertise that they bring to the table.

The remaining two elements of the design fundamentals are, first, that you should emphasize relationship management. You should stress that this is a partnership. It's not IT versus actuaries, actuaries versus IT. This isn't the Super Bowl; we all have a common goal. And, lastly, you should modify your management measurement. You should delegate rather than dictate, and you should facilitate communication. Often as actuaries we're not involved in the hands-on doing. We might have students who work for us who are involved in working with IT or, vice versa, the IT area has delegated a lot of the preliminary programming to students straight out of college.

So as an actuary, it's your responsibility to delegate rather than dictate. I am not an expert on software systems. I just define requirements for what a system should have. Therefore I am delegating. I am not telling them how they have to do something, that they have to write it in C or they have to do it in dot-NET. I want to encourage them to choose what best meets my needs. That gets back to facilitating communication. We should go back and forth. We should develop cycles to help improve communication.

For the construction fundamentals, and this again is more from the IT side of things, focus should be on reducing complexity and providing automation. As a programmer, you should anticipate diversity. You should construct your programs

so that they're structured for validation, so that you can easily test that the system does what it is supposed to do. You should follow established standards. There are standards for programming languages,. which are very important to follow.

Now I'm going to go through a couple of software development methodologies. These methodologies are mechanisms that I would encourage you to consider using when you're looking at a large overhaul of a system or you're anticipating using a software system as a means of solving a particular problem that you might have. A lot of the software development methodologies originated in civil and mechanical engineering. They were designed to minimize the use of, in time, very expensive computing resources. Many of the key issues are amenable to mathematical analysis versus reliance on peer review.

These older methodologies, the predictive methodologies, are better suited for projects that have a very small design element relative to the construction part. If you think about it in terms of the history of civil and mechanical engineering, a lot of these methodologies evolved because very high-priced engineers were constructing things in their minds of what a bridge should look like, and then you had many laborers actually building the bridge. So these old methodologies were very good for building bridges, but they're not very good for building software.

Some examples of predictive software development methodology are the sequential V, the waterfall and the Boehm spiral model. I don't really want to go into these in a lot of detail, but I want to give you an idea of the methodologies out there that you could implement in your various projects that might improve the result at the back end. The traditional waterfall method starts at the top, and then it gets handed off, and it kind of trickles down, trickles down, trickles down.

So you're going to document the concept, you're going to identify requirements and analyze them, you're going to go through some architectural design. Then you're going to go into more detailed design, and you're actually going to hit the part where you do the coding. Then you're going to integrate everything, pull it back together and deploy it to the end user. So this is a traditional method. It starts in the center: it starts spiraling out, and it's more of a cyclical approach. You start off small, and then you grow and grow.

Some of the more current software development methodologies are called adaptive or agile. They evolved due to the need to develop in Internet time, meaning I have to get something to the Internet today. I can't take months and months designing something, or months and months testing something. So those old process flows really don't address the desire to get something to market faster.

You will have working code much faster than you would, say, under some of the older predictive methods. It can be better suited for projects that have a larger design element in them rather than a construction element. These methods don't work very well for really large development teams. Many of the examples I'm going

to give are geared toward highly motivated, highly efficient teams of eight to 10 people, not these large 200-staff projects.

Some of the examples are XP, Cockburn's Crystal Family, Open Source, Highsmith's Adaptive Software Development. There are a lot on the Web today where you can go out and find more information. The idea behind extreme programming is that you will have the business client who will define requirements, and they will participate in release planning. Then we will have iteration planning. Note that this is something sequential. Unlike the waterfall where you start at one end and you go all the way through to the other end, this one is constantly cycling back. You have lots of teeny, tiny releases in between so that you have progress shown throughout the development cycle.

I find that this works very well when we're bringing consultants in to get the properties of a particular product onto our variety of systems, whether it's the valuation system, the modeling system, the illustration actuary system or a system that the agents in the field use, where it's new, so we're not quite sure the design that we're going to want in the end. Instead, we'll just take baby steps, and we'll kind of go around and around without a big target deadline, but we'll set smaller intermediate deadlines. So you've got small releases. The agile folks refer to that cycling back as refractoring, where you make a small change to your code to support the new requirements.

Another strong item of the XP and the other adaptive methods is that you have adopted a very simple design. It should be very easy to understand so that at the same time it's very easy to make changes to the code down the road. Collective code ownership means that no single person owns the code; you all share in it. Any particular programmer should be able to work in any part of the code at any part of time. Some of the legacy systems out there have people who say, "I write Part A, and that's all I work on," and, if you ask them a question about Part C, they say, "That's not my job." Well, that isn't true in the adaptive methodologies of software development. Everybody owns the code; everyone is responsible.

For the onsite customer, another important point of this methodology if that you have continuous access to the other people. That's why I use this with the consultants a lot. They're in our shop, or we're in their shop. We're working together as a team going back and forth. It doesn't work really well when your vendor is overseas, for example, and you have access to them only while you're sleeping. So you really need that integrated approach for success in a development environment.

Coding standards: one of my soapboxes is following established standards. It makes it much easier for other people to pick up after you. Let's say you leave the company, or the programmer leaves the company. If you followed established standards, then it's going to be easier to transition maintenance of that code over to someone else.

Scrum is another method that I like to use. This is getting back to that team approach.Scrum is where you've got a group of people, and they kind of cluster around. Think of it like a soccer team of four-year-olds, where they just all run to the ball. They don't know their positions. And gradually it moves downfield, so that they start one place, and they ultimately reach the goal, but they may have drifted a little bit right for a while, and they may have drifted a little bit left for a while.

Those were some of the elements of design. Now I'm going to go over some of the construction process elements. Remember, my perspective is from IT, so I'm trying to educate you on some of the processes that IT uses. Those last methods focused on how they design software systems. Now we're going to talk about how they construct software systems. The specifications document is a comprehensive statement of the problem that you're trying to address with a software tool.

In the analysis phase, the problem is laid out in a form that will lend itself to an arithmetic or logical analysis. Then the programmer or the person involved in the construction process creates a flow diagram that shows all the operations and the sequence in which items are supposed to occur. The encoding is the actual coding part, which converts the operations into computer language. This is where Emil referred to his compiler for Excel, where someone has given him the specifications, they've done some analysis, they've determined how things are supposed to occur, and then they just push a button, and it automatically creates the computer language. Debugging is the process of locating errors, and then documentation. Emil gave an example, although briefly, of how you can actually even automate elements of your documentation phase.

There are three different types of construction styles. There are linguistic methods, formal methods and visual methods. I am not a computer engineer, so I am not going into these in any detail. But you can talk to a computer engineer, and I'm sure they can give you loads of information on these three methods. Some of my observations from having worked with IT for the past 11 to 12 years is that if you're dissatisfied with your results, then maybe your company or your business unit is using an inappropriate development methodology, or maybe you're using an inappropriate construction style. Maybe you're using a visual method, and you should be using a linguistic model, or, similarly, you've adopted an old predictive method for something you want, but you want rapid turnaround. So maybe you should employ some of the adaptive methodologies.

Do all of the parties know their roles? As an actuary, do you know what your role is in that methodology process? Does IT know their role? Do you have the correct skill set? If you are a programmer from a legacy system, you are not going to work particularly well in an adaptive or agile-type environment, whereas a student right out of college is going to be up on all of the new technology and will fit well in that type of environment. But they aren't going to be as well suited for a legacy-type environment or for working in a predictive-type methodology system. I can't stress

the importance of communication. Those feedback loops that the adaptive or agile methods incorporate into the software development process, I think, are a big plus.

So here are some reminders. Are you guilty of any of the following? When you are designing your specifications or when you're testing the systems that the actuaries have given you, have you introduced noise? Noise is where you are providing information to me that has no relevance to the task at hand. Or silence: I get this frequently when there is a feature of a particular product and the person designing the specifications hasn't told me about it. Overspecification: have you actually included in your specification a way to solve the problem? I'm the programmer; that's my choice. If I choose C or dot-NET, or I choose Excel with a Visual Basic macro in it, that's not your call. Your call is to design for me something I can program from. It is my choice as to what language I want to put it in.

Contradiction: contradiction is where in your specification you actually give two different interpretations of the same thing. You say a plus b equals c, but later on you say a minus b equals c. Don't provide me with contradictions. Ambiguity: there is a lot of ambiguity in the specifications that I receive from actuaries. Maybe they haven't been particularly specific about how some feature should work or there are two different ways that the statement could be interpreted, kind of a to-may-to/to-mah-to type of thing, two different ways of saying the same thing. Forward reference: in your specification document have you started talking about a feature that you haven't even defined yet? And the last one is wishful thinking: that also gets into your trying to tell me how to code something and how to implement a resolution. That is my choice as the programmer.

Just for the flip side, because I'm an actuary, even though I'm taking the IT side, I also wanted to point some fingers at IT and say, Hey, guys, when you're doing these things, make sure you design a program that's portable, flexible, complete and comprehensive, dynamic and responsive, consistent, clear and concise, simple, and that provides suitable controls. So if IT meets all of these requirements that I have, and I meet all of their requirements, then hopefully we'll come together, and we'll have less discord between the two parties.

MR. HARTFIELD: You may not recognize the face of our next speaker, but I'm sure you recognize his work: Frank Reynolds. He's celebrating his 40th anniversary in the Society, and he's published over 130 Actex manuals in his career. So, without further ado, here's Frank.

MR. FRANK G. REYNOLDS: I'm afraid I go back some years in the computer business. Those of you who have come here to better learn how to soap a program, I'm sorry, I'm not going to get into these technical details. How many of you know how to soap a program? Compile? It's a more modern word for the same thing. I go back quite a long way. I've worked on the IT section side running a computer installation, I've worked on the other side in the line areas. My job today is to provide the comic relief, namely, examples of what can go wrong.

The first case: a systems analyst was called down to the controller's office. A new system had been introduced. In the conversion process they discovered one life who is 102 years old. The system provided for a two-digit age field. The controller demanded that all the programs in the system—and they were looking at about 600 different programs at that time—be changed to accommodate the three-digit age of this particular person. The systems analyst went back to his office for a while and decided that the cost was going to run somewhere in the neighborhood of about $300,000. The next question he had for the controller was "How big is this policy?" "It's a $1,000 policy. Have you got it all fixed up yet?" The systems analyst said, "Well, I've got an idea. Why don't you have the agent, the branch manager, maybe the director of agencies give a great big bash at the retirement home where this man is living, call in the local press, and present him with a check for the amount of the policy, which we're holding over a $1,000 reserve for, I understand, and say that this man has outlived the mortality table." Well, after some argument the controller decided that maybe there was some merit in this idea. They did it. They took $1,000 off the books as a liability and saved themselves $300,000 in programming costs. The point I'm trying to make is this: they got more than $1,000 worth of publicity out of this. It was a much cheaper way to do the same thing. That's one of the things to consider when you're doing systems work; think about it.

In another example, the actuarial and agency departments turned around and announced a whole string of new products on December 1. This was the first the computer systems department had heard of it. The systems analysts were a little alarmed at one thing: there was a new, different policy fee for the individual health policies. The head of systems had to go to the chief actuary and say, "Um, er, uh, we've got a small problem. It's going to take us six to eight weeks to program this new innovation, and, um, er, uh, for your end you're going to have to value all of these policies manually." The roof was raised about four stories, and there was an explosion.

The next step was that the head of the systems division walked in one day, and here was his desk covered about so deep in books. The actuarial department had presented him with every project that they had ever thought of. He had to go return the books to the actuarial division and explain that maybe the solution was that when they started planning a new rate book revision, about three months before they implemented it, they involve the systems department. Then maybe both parties could work together. The systems would be ready on time, and the actuarial people would get what they wanted. The point I'm trying to make is, you need communication, and you need it at the right time.

The third case involved a reinsurance product. One of the people in the reinsurance division got transferred to the computer area. Five or six years later, the company needed a new reinsurance system, and they assigned this man to do it. Well, he'd worked for 15 years in reinsurance, and he knew all about reinsurance. So he sat at his desk without saying anything to the actuarial division that was involved and

designed a new system, then he wandered into the head of reinsurance's office one day, and said, "Here's the system. It's Monday morning; I need it approved by Friday afternoon so we can start programming next week, which I've got all arranged."

The reinsurance people said, "Um, er, uh, there have been a few changes since you were here some years ago. What's going on?" They started checking. A whole bunch of new management reports that were needed were not provided by the system. It was weeks before they managed to get everything straightened out. There was a grand fight between the two division heads, and I mean a *grand* fight. The systems chief was incensed and then absolutely humiliated when he found out the system wasn't going to produce. Moral: Sometimes the systems people try to design something, and they think they know best. Sometimes they don't, and they should involve other areas.

The next case is a cute one. I was involved in this particular case. We were asked to do a group annuity valuation system. So I asked, "Well, what's normal retirement age?" "Sixty-five." "Well, what's the oldest that we have people retiring?" "Well, some people go into their 70s." I ran around the bush on this one with the line area for several hours and decided, "Okay, there's probably nobody over age 80." So I actually programmed the computer assuming that there would be nobody still with an active working life contributing to their group annuity at age 92. The system was tested; it went through. December 30, 2:14 a.m., I got a telephone call: "Frank, your system's blowing up." At that time I was working in Winnipeg, Manitoba. As some of you know, Christmastime out there can be very cold, like 25 to 30 below, and this was a windy night. I had to get out of a nice, warm bed, from under a nice, comfortable eiderdown, and go downtown. I discovered there was somebody in the system who was 94 years of age. I figured, well, somebody's transposed the digits, it's 49, we've got to pass that record. I stayed in my office for the next hour and a half till it went through. The next morning I went downstairs and said, "There's a coding error here." " Oh, no! This is Mr. Fredericksen. He *is* 94, and he's actively saving for his retirement. He hopes to retire before he's 100." I'd made the mistake of not making them go through the file and getting the extreme case. I paid for it.

In another case that I saw, the systems analyst went to the head of the systems area and said, "We've got a problem. We're trying to bring in a new valuation system, and the guy who runs the valuation department wants us to incorporate an approximation for calculating the reserves and accidental death benefit, and it involves a huge spreadsheet." Well, the head of systems thought the guy was exaggerating. They went over to the valuation head, and out comes a sheet. The guy started over in that corner, and it came roughly over to about here. They wanted it programmed this way. The suggestion was made that doing it accurately would be a little bit cheaper. "No, this is the way the chief actuary's done it, and this is the way we're going to do it. We've done it this way for 40 years, and we're going to do it for another 40. Program it."

The systems analysts and the systems head sat down and figured out that it was going to cost about $300,000 to do it that way, versus $50,000 to do it accurately. They took it down to the chief actuary, the chief actuary listened very patiently, asked a lot of questions, and finally agreed to do it accurately. Moral? Sometimes approximations are useful, but sometimes it's a lot easier and a lot cheaper to do it completely accurately.

Another case goes back to before most of you were born, the time when they implemented the Canada Pension Plan in 1965. At that time, the Canada Pension Plan was announced early in the year for implementation on January 1, 1965. So, early in 1964, I was asked to put together a quotation system that would generate quotations on integration. Until that point, pensions had been the same regardless of your income. Starting at that point there was going to be integration with the Canada Pension Plan both on contributions and on benefits. So we needed a system. I had six weeks to get this system out so that the people could show the quotations in September and get everything back in before Christmastime.

My first question was "How do you calculate your premia on the group annuity business?" "Well, we want to change our rate basis. We're going to change it as part of this process." "Fine. But if I'm going to get this programmed, I need to know the general outline of how you're going to do it. Are you going to have a policy fee plus rates? Are you going to use discounts varying by size as you now are? How are you going to do it?" "We don't know. You just program, and we'll tell you after we develop the system 10 weeks from now for rates what we're going to do." "But you want this system in six weeks." "Yes." "But you're not going to give me the rate basis till 10 weeks from now. How can I do one before the other?" "We don't know. That's your job." They were trying a lot, too much at once. And this is something that you'll often find, that people don't realize if I had known it was a policy plus p plus rate basis, I could program that, but unless I knew the general form of it, I didn't have a hope. And it caused a lot of comment.

Another case that I was involved with was on the intercompany disability study in Canada. I was sitting in Waterloo, Ontario, and the compiling company was 1,250 miles away. I agreed and signed off on the basic specifications. They were programmed, tested, and the first of May two boxes of paper arrived on my desk: "Please check out and get it back to us by the 15th of May." Well, I was in the business of giving seminars at that time for the university as well as privately. My busy time starts about the first of May when the Society publishes its education catalog and ends sometime late in October when I finish giving my university seminars. Now, that particular year there was a change in the exams, and the university seminars I was preparing were rather extensive, shall we say?

I phoned them back and wrote them and said, "Look, I can't get to this till the new year." "What? You're sitting doing nothing." I got through it in early January and gave them a list of some 30 different errors that I found. One of them was the cause of disablement: it was a blank sheet of paper, except for the cause down the

side and the rows of headings across the top. The rest of the sheet was blank. Well, I sent that out in January. This had to go into the computer system's scheduling process. So it got scheduled for a year and a half to two years later. First of May again, yes, you guessed it. I got the stuff again. I got it out the following January, and this sheet that had been blank before now had 100 percent in every field. I knew that couldn't be, that everyone was just exactly 100 percent of what was expected. So I said there was something wrong with this. And the errors had gone down; they had, by and large, corrected things. I think there were only two that were still mistakes, but they had uncovered maybe eight or 10 new things.

This cycle went on several times till finally the system was abandoned. Why? Simply because the systems department didn't realize that the operating area, me, had a cycle. There were certain times that were busy, and we couldn't turn around and produce. Similarly, instead of scheduling, knowing that there was going to be a time when this material was going to come in and putting in a schedule, they waited till they got the results back and then put it into their scheduling hopper, which was about an 18–24 month cycle. You can't have this sort of thing, or a system never gets up and run.

The next case involved a system I was developing for group annuities and a billing system that involved consolidated functions. One of the people working for me at the time was a young woman with a Grade 11 education who had worked in the group department at one point. We developed all the forms in English, sent them to French translators because in Canada we need the forms in both English and French, and the French translators translated them all into French. I'm not trying to deprecate them in any way, but most of them barely read English, let alone spoke it, and they certainly didn't speak group annuity.

The forms came back, and Jeannette looked at this stuff, the woman with a Grade 11 education, and she said, "Frank, do you mind if I try translating it?" I thought for a moment and said, "Hmmm, she's absolutely fluent in French, she speaks group annuity, and she speaks English." I let her do it without saying anything to anybody. The system went out. The forms got all printed, went out to the field force, and there was one great, grand gafuffle. This was first time that the company had ever produced forms in French that were intelligible.

The woman involved was Jeannette Davignon. She eventually turned around and went back to university and ended up as a Fellow of the Society of Actuaries. My moral here is that she appeared to be a lot less qualified than the French translation department, but because she spoke both languages fluently, plus group annuity, she was far more qualified than our translation people. Use your best source possible.

Another system I saw involved group annuities, a big consolidated function system, one that we referred to as a legacy system, a great, big system that took man-years to do the systems work and man-years and man-years to program.

Unfortunately it contained a flaw. The reserves were calculated correctly, the administration was correct, the billings went out correctly, but the outstanding premia for accounting and actuarial purposes were incorrect. Three people in the actuarial division had to turn around and work full time trying to figure out what the reserves should be for year-end and to check them.

Somebody from the actuarial division went to the systems division and said, "Could we have a look at the programs because, hey, there's something wrong here?" "No, that's our job." "Would you let us submit some test data and see if we can figure out what it is?" "No, that's not your job. It's the line area and ours," and they signed off. Five years later, the system finally had to be abandoned at a cost of around $5 million. Why? Simply because somebody was guarding their turf a little too much.

Another example was in a company where a consolidated functions system was brought in. Unfortunately, they didn't have the manpower to bring it all in at once, and the valuation was left on the old system. The record-keeping was done in parallel, and the accounting was on the new system. Well, the accountants said the new system would produce results as of December 31. The cutoff would be on December 31. Then the actuarial division said, "Look, we're on the old system. We've got to cut off on the 15th because there's a lot of manual work in the old system." The accountants said, "We don't care."

The result was that on the third of January, when the results came in, everybody got called down to the president's office. There was one serious problem. The company had no earnings. Guess what? Cutting off the two systems on two different dates had resulted in an error the size of the amount of earnings. So for goodness sakes, make sure all the parties, all the line areas and the systems areas, work together and know what each other is doing.

There is a lot of discussion about the fact that systems and actuarial are different. I come from a university that is noted for both its actuarial and computer programs, and I have some records in my office. What we used to do back in the 1960s at my university was check students and give them the actuarial aptitude test. The ones who passed were allowed to go on in actuarial science. The ones who failed were put into computer science.

This story goes back to about that time. There was an actuary working for the company who wanted to develop a cash value disability product. Those of you who are familiar with that type of product know that there is a tremendous dependence of the premium on the cash value and the cash value on the premium. The two are very closely intertwined. So one of the top systems analysts was assigned to this, somebody with a master's degree from the University of Waterloo in computer science, really up there with the latest technology. He programmed things, and it was all checked out. On Saturday morning, he started a mainframe computer at 8 a.m. Well, about 2:30 the next morning, he got a phone call from the computer

staff. They wanted to go home. Why? Because there was nothing else left to do, but his program was still running. He had them stop it. He discovered that it had narrowed the premium down to somewhere between $40 and $140. On Monday morning, he got called to the chief actuary's office, an old gentleman about my age, who asked what the results were. "Well, somewhere between $40 and $140, sir."

The old gentleman looked out his window at the beautiful Manitoba maple that was in green leaf at the time and thought for about a minute and a half and asked, "What did you use for a starting value?" "Zero." That's perfect, good computerese. That's where you start. That's what they teach us in university. He looked outside for another 15 seconds and then said, "Try $85 for your starting value." Next Saturday morning, 8:00, the program gets started with $85. Two minutes and 15 seconds later, the program stops. The systems analyst gets called in. The program had run, and the premium was something like $84.27. Moral? Communicate.

I work in a shop where theory is something that we dispense and teach students. When I started, most of us were practicing actuaries and not theoretically oriented. I was the theory man at the time when I started. Now I find I'm the guy who's practical, and the others are more theoretical than I am. Realize something: some of the old goats like me who have been around forever can turn around and, using rules of thumb, come fairly close to the approximations that can be done with all these technical things. I noticed one of the older, gray-haired people in the audience who has got a little bit of experience going, "Mmmm, I know what you mean." Use that experience that's around, because sometimes it's worth a lot more than technology.

Finally, I saw one computer system where the line area and the systems people had agreed on 250 consistency checks. Some months later, the systems analyst was down in the line area, and in stumbled somebody with a whole pile of error messages printed out, thousands of them. The systems analyst asked, "What's going on?" Well, it turned out that three of the consistency checks that had been asked for worked beautifully. The only problem was that most of the policies violated these consistency checks. So every month when things were run, they got almost a complete list of their policies. Instead of telling the systems people that, "Hey, the consistency checks that you're running, some of them are worthless, let's take them out," they just turned around, let them be produced, and then threw them out and, in fact, were ridiculing the systems people for all their mistakes. Communicate your problems and make sure that any error messages that you're using are not only technically correct, but that they're actually useful. Thanks very much.