TECHNOLOGY SECTION

"A KNOWLEDGE COMMUNITY FOR THE SOCIETY OF ACTUARIES"

CompAct

CompAct Electronic Newsletter • Issue No. 24 • July 2007 • Published in Schaumburg, Ill. by the Society of Actuaries

Inside

Actuaries
Risk is Opportunity.℠

Technology Section Newsletter
Issue Number 24
July 2007

Published quarterly by the Technology Section of the Society of Actuaries

475 N. Martingale Road, Suite 600
Schaumburg, IL 60173
phone: 847.706.3500
fax: 847.706.3599

World Wide Web: www.soa.org

Letter from the Chair

by Paula M. Hodges

We are coming upon a very important time of the year—our board and council elections! The candidates have been recruited, the ballots drawn; now it's your turn to vote!

But … maybe you had been considering putting your name on the ballot, and somehow didn't get it done. Maybe you're considering running for council next year. Well, it's never too late—or too early—to get involved. Our council met last October, intent on building opportunities for networking, improving communications, enabling information sharing, providing publications and enhancing technology education. We have so much we can do, but we can only succeed if we have more volunteers from our general section membership. So, we'd love to get you involved now! It would be an excellent "campaign" for your election to the council in 2008.

If you don't know how to get involved, you can read our TechUpdate e-mail. We send it out every two months and it has lots of information on the activities of the section. We send the update to let you know what your section is doing on your behalf, and also letting you know where we need help. If you would like to review the latest editions, you can find the TechUpdate e-mails on our Web site at http://soa.org/news-and-publications/newsletters/technology/pub-technology-section-updates-details.aspx. Or, just call one of the council members to see where you'd find the best volunteer opportunity for you.

I hope you take me up on this volunteer opportunity. The experiences of participating actively in the Technology Council have been extremely rewarding to me in so many ways. Participating actively in a section of the SOA provides exposure to so much more than just your section's interests. By getting involved, doors will start opening up to you, in ways you never dreamed possible. I strongly encourage you to become involved. Then, when the time is right, put your own name on that council ballot.

Paula M. Hodges, FSA, MAAA, is senior manager-product management with Allstate Financial in Lincoln, NE. She can be reached at Paula.Hodges@allstate.com

Good luck to this year's candidates and welcome to this year's new volunteers!

Paula Hodges
Chair - Technology Council

Scenario File Format Project Update

by Carl J. Nauman

Carl J. Nauman, ASA, is a consulting actuary with GGY AXIS in Toronto, Ontario. He can be reached at Carl.Nauman@ggy.com.

The Standard Scenario Format Working Group is continuing to develop a tool for actuaries to store and exchange economic scenarios using a standard format. The last update can be found in the January 2007 issue of *CompAct*.

We now have a tentative XML format for scenarios. Some minor adjustments may need to be made during the development of the utility program. It will provide for yield curves at various bond qualities, equity returns and other economic indicators, all for multiple countries.

Our main focus over last few months has been the utility application. Initially, it will be for reading, writing and viewing the scenarios. The first phase of the development is nearing completion at which time we will begin testing.

We also have been writing documentation for the standard format and the utility. To date, progress has been steady and the project is proceeding smoothly.

Book Review:
Why We Think That We Think

by Carol A. Marler

Douglas Hofstadter's 1979 book, "Gödel, Escher, Bach: An Eternal Golden Braid," won a well-deserved Pulitzer Prize. In it, with youthful exuberance, the author displayed a wide assortment of paradoxes, self-references and symmetries that exist within and among the fields of music, mathematics, literature and art.

Twenty-eight years later, an older, sadder and perhaps wiser Douglas Hofstadter again tackles the difficult questions of systems so complex that they are able not only to refer back to themselves but also to analyze what they themselves may mean. In "I Am A Strange Loop," Hofstadter again explores the meaning of Gödel's proof, while also considering the implications of consciousness, his academic area of specialization.

The book begins with a reflection on Hofstadter's choice to become a vegetarian, and how his guidelines for what not to eat have evolved over time. He uses this meditation to explore the idea that consciousness is not a binary phenomenon, but rather has fuzzy boundaries. Some conscious beings, such as humans, clearly have a more extensive repertoire of consciousness than those animals commonly considered to be "lower" in some way, such as dogs, cows, goldfish, snails and mosquitoes.

As before, he takes the reader through an explanation of Gödel's proof which requires no sophisticated mathematical knowledge. He then goes on to conclude that any sufficiently complex self-referencing system has the capability of being conscious. In fact, he concludes that this emergent property, consciousness, arises automatically from the fact of its use of self-referencing symbols. Such a conclusion is not new, and may not even be controversial … until it is applied to inanimate systems.

Part of his book works hard to debunk the theories of those, such as John Searle, who believe that only organic beings have the "right stuff" to attain consciousness. One of his most telling counter-arguments is the audience reaction to characters such as R2D2 and C3PO, which are clearly mechanical, but are readily accepted, in context, as thinking, conscious beings. In part, his argument for why we feel this way is that these cute robots exhibit emotions. When they look and act fearful, we do not say to ourselves that they are merely programmed to react in that way; we simply go to the idea that their behavior (as much as same behavior by humans) means that they in fact are fearful.

Personally, I am not convinced that is it simply a matter of displaying emotion. A key element in storytelling is that the audience will care about the subjects of the story because of the way in which their experience produces change in them. The mere fact that these robots show fear or other emotions is not enough; the audience relates to them as they respond to the fearful situation, for example, by overcoming their fear and taking necessary action (to advance the storyline). To me, this change must be the result of a choice made by the protagonist.

And here is where the author and I part company. Hofstadter concludes that his explanation of consciousness has no room for free will. Whatever we do is an outcome of our innate desires. If there is a conflict, one or the other desire is stronger and thus is certain to be acted upon. If we change, you see, it is because we want to change. It's all presented in a deterministic context. I'm as uncomfortable with his rejection of free will as he is dismissive of the idea that will could ever be "free."

His argument in part involves a rejection of dualism—that there is something to a human "mind" other than the brain and its symbols, including the self-referencing symbols of "self" and "mind."

In addressing this whole issue, he finds it necessary to consider the activities of a brain at two levels. One level, the symbolic, is driven by the underlying molecular elements of brain chemistry (which he, unfortunately does not really explore) and the fact that such molecular reactions are actually mediated by subatomic particles which behave in a manner both probabilistic and boundaryless. The probabilistic element may be familiar to actuaries (at least those who have studied financial economics) as the equations of Brownian motion. The boundaryless feature, at the quantum level, means that any subatomic particle can be influenced by other subatomic particles regardless of distance.

In his view, all that happens at the "lower" level is essentially meaningless. Nevertheless, this subatomic particle soup is what drives all the activity at the "higher," symbolic level, where our thoughts and perceptions take place. I could have wished for a closer look at some of the intermediate levels, where much research is currently being done on both the biochemical aspects of neural activity and the physiological structures of nerves and brains. Maybe this will appear in another book.

Along with free will, of course, Hofstadter also rejects the notion of a separate "soul" or "spirit" which can continue in existence in the absence of the physical structure of the brain. How, then, does Hofstadter deal with the tragedy of his wife's premature death, when their older child is only five years old and the younger one is two? His philosophy regarding this is explored at some length in Chapter 16, which is made up of excerpts from some e-mails he wrote to a good friend and fellow consciousness researcher, Daniel Dennett as he came to grips with this personal tragedy. The ideas expressed are actually those that Hofstadter had considered even before his family was touched by this death. What he concludes is that the personality of an individual is carried in simplified form in the mind of those who knew her (or him), and that so long as those concepts remain in memory, the person still exists. His philosophy is not new—it is explored in literature of at least 40 years ago, such as a short novel by Romaine Gary which I read as a college student.

"Part of his book works hard to debunk the theories of those, such as John Searle, who believe that only organic beings have the "right stuff" to attain consciousness."

Hofstadter also considers another loopy idea. Self-awareness includes, of course, awareness of the fact of self-awareness. Since he believes that there is no separate "being" to

(continued on page 6)

continued from page 5

"Along with free will, of course, Hofstadter also rejects the notion of a separate "soul" or "spirit" which can continue in existence in the absence of the physical structure of the brain."

experience this awareness, he somehow concluded that the whole thing is a kind of mental illusion. Through the magic of perception, Hofstadter sees us as mirages who perceive ourselves. As he expresses the idea in his concluding section, "In the end, we self-perceiving self-inventing locked-in mirages are little miracles of self-reference."

For the reader seeking parallels with the earlier book, there are certain playful insights and gems of self-reference to be found. For instance, many chapters begin with an illustration of something forming a loop. One of my favorites features a grinning Douglas Hofstadter and 10 other people who are arranged in a circle facing clockwise. Each person is sitting on the lap of the person behind him or her.

In support of his idea that our self-perception is a mirage, he presents an illusion that he once experienced with a box of envelopes. Grasping them all in order to take them out of the box, he became convinced that they contained a marble somewhere within. After examining each envelope, and finding no marble, he realized that the feeling of something solid in the middle was the result of multiple copies of the triple point where envelope flaps are glued together. Even knowing that this was what caused the feeling, he still had the distinct impression of something hard and round.

He recounts this experience in a lecture, and one of the people in the audience is so taken with the image that she goes home and writes a poem about the illusion. The poem makes its way into Hofstadter's hands, and he (with permission of the author) includes the poem on page 94.

Footnotes, as it happens, are located in the back of the book. On page 376, a penultimate footnote refers to a certain literary device used on page 361. The form is called paeonic meter and consists of 40 repetitions of the basic "foot" which is simply three unstressed syllables followed by a stressed fourth symbol. The two paragraphs on page 361, referenced by the footnote are in this form … so is the footnote, and also the final footnote, which refers to page 376, and in particular that previous footnote.

This book reiterates the basic concepts of Gödel's proof, and gives a bit of the mathematical background. About a century ago, Alfred North Whitehead and Bertrand Russell set about to establish a completely rigorous foundation for mathematics, in which all the logical principles used for basic arithmetic are set forth and the operations of addition are derived as theorems therein. The book they published was called "Principia Mathematica(PM)." Their purpose was to assure users of mathematics that the structure was secure—that it was a complete system and that it was internally consistent. What they did not realize was that completeness and consistency are mutually incompatible, just as Heisenberg's uncertainty principle tells us that it is impossible to know exactly both the location and the motion of an "elementary" particle.

What Gödel accomplished was to show that the logic of PM could also be applied, not just to basic mathematics, but also to PM itself. By high-level logic, he constructed a way of making a theorem that essentially declared, "This theorem cannot be proved within the system of PM." To quote Tevye, "Sounds crazy, no?" The key implication of this proof for mathematicians is that mathematical statements exist which are true, but which cannot be proved. And, of course, it is impossible to know which unproved statements might actually be the unprovable ones.

Like Hofstadter, I enjoy the alternate formulation of the theorem, which is worded only slightly differently, "I cannot be proved within PM." Can a theorem truly speak in the first person? How many of us, kicking off a computer program that will run for some time, refer to it as "looking" for the answer to the problem? Do we ever speak of an Excel goal seek operation as "wanting" to find the number that fits our criteria? I am not suggesting that there are computers today that are conscious. No, the degree of complexity and self reference has probably not yet been achieved. But it is interesting to reflect on just what sort of advance might allow us to construct a mechanical entity that really does think, reason and even recognize itself as a being that thinks and reasons.

If such speculations seem airy and meaningless to you, then there is no reason to read this book. But, if like me, you have often reflected on the meaning of consciousness and how it relates to complexity and self-reference, you will want to add this book to your collection. You can put it next to Lewis Carroll's *Alice in Wonderland*, and to *Gödel, Escher and Bach*.

Carol A. Marler, FSA MAAA, is associate actuary at Employers Reassurance Corporation, Indianapolis IN. She can be reached at carol.marler@ge.com.

Program Well and Live

by Mary Pat Campbell

The End is Nigh!

In looking for articles on actuarial programming habits, I came across "Document or Die," by Jim Toole from the February 1995 issue of *CompAct*. "That's a bit harsh," I thought, noting from my own experience that documentation was pretty thin on the ground, and yet plenty of computing was getting done. If nothing else, dead programs weren't littering our directories. Just as kids ignore parents after excessive warnings of "You'll poke your eye out!" and "Your face will freeze that way!" the lack of calamity in general means most actuaries will ignore the exhortation "Document or die."

My actuarial modeling experience (all four years of it) has been that if I write a program or some code for Excel, the primary user is going to be me. If I'm documenting my code, the likely audience for said documentation is me, some months after I had last changed the code. I don't particularly want or need to step through paragraphs of documentation within my own code. Generally, if I am picking up a program again, it is because I am changing the code to use it for another purpose, or updating it for another year's worth of experience. It takes very little documentation to achieve this, and I will show how and why here.

This article is for those who have to do a lot of custom programming, such as with modeling, and for those who are managing actuarial students doing serious numerical programming for the first time in their lives.

Goals of Actuarial Programming

- In numerical programming I have various goals to attend to: The program calculates what it's supposed to.
- The program does its job in as short amount of time as possible, using the fewest possible resources.
- The program is easy to modify.
- The program has "mobile code."

The first goal is the sine qua non. If the code doesn't do its primary task, all is for naught. This usually is the easiest part of programming—many with little programming knowledge and experience can accomplish this goal. As that's the immediate goal of any programming task—and few want to read other people's code—most are not required to go beyond this goal.

The second goal—optimizing the runtime or memory use—is more difficult, and requires more knowledge of compilers, algorithms and the like. Sometimes you can optimize your problem before it even hits the computing stage (such as scenario reduction in Monte Carlo modeling for ALM). It's usually a good idea to take some formal courses in numerical algorithms or computer science to learn this, which is not something that's easily conveyed in a short article. As well, with the spread of distributed computing, some might not even have to optimize much as long as they've got hundreds of idle PCs available.

The last two goals—making the code easily modifiable and portable—are what makes your future jobs easier, and what this article is about. You don't want to spend time coding if you already have perfectly good code from an old problem that can be copied for current use. Also, if your original problem changes, and you have to alter the code, you don't want to be spending time trying to hash out what you did the first time; you might as well program from scratch.

If you cultivate good programming habits, you can spend more time analyzing and optimizing, and less time coding and debugging. Following are some tips I have come across and have tried incorporating into my own code.

Tips for Good Numerical Programming

1. Organize your programming flow well.

This will go a long way in preventing problems and will help with achieving the first two goals of efficient and working code.

You want to avoid the dreaded disease of "spaghetti code" where you have gone off programming before thinking through the logical flow and parts of solving your problem. I've seen code where variables are declared and initialized in the middle of the code right before they were used; I've boggled at incomprehensible or redundant logical tests; and I've gritted my teeth at loops with escapes built deep inside (dangerous!). If such code ever does work correctly, it is unalterable. Try to change one line and the whole structure can fall apart.

"In short, simply name your variables according to what they are, the procedures as to what they do and functions as to what they return."

So think before you code. You may not need to use a flowchart, but you do need to have an overall flow of what is calculated when, and it's also best to group together steps by category. For example, even if files are used much later in the code, I usually open all of them at the very beginning and initialize all variables at the start. That way, if I have to remove or add a file or variable, I can do that easily.

2. Create self-documenting code with good style and naming conventions.

The first time I wrote a program with three nested loops, I used the names "Hickory," "Dickory" and "Dock" for my loop indexing variables—very cutesy. My excuse: I was 12 years old. (I went on to use "Rikki," "Tikki," "Tavi," "Bibbity," "Bobbety," "Boo," and "Helter", "Skelter." It took me a while to figure out why I couldn't use "snap," "crackle" and "pop," which goes to show the danger of using regular words as variable names.)

These variable names told nothing of what they were looping through; at best, I could tell the outer loop, the middle loop and the inmost loop. It's all very well to have non-descriptive variable names for a school

(continued on page 10)

assignment, but it becomes untenable when you have real problems to solve. Suppose you have to loop through age, issue year and settlement year cells when projecting liabilities. In which code would it be easier to confuse the looping variables: the one with variables i, j and k, or the one using variables AgeIdx, IssueYrIdx and SettleYrIdx?

On top of my bad variable name habits, I would name procedures thusly: "DoStuff," "DoMoreStuff," and "DoStuffThree_TheReckoning." Again, very cute, but not very enlightening. To be sure, I dutifully commented the procedures as to what they did and what inputs they took, but any time I had to use one of them, I'd have to look it up to see which one I wanted.

"The first goal is the sine qua non. If the code doesn't do its primary task, all is for naught."

In short, simply name your variables according to what they are, the procedures as to what they do and functions as to what they return. Some examples of names I've used follow:

RowIdx *(loop variable for looping through rows)*
IntPayt *(variable that will take interest payment)*
CFRow(Year, Month, Day) *(function that returns the row a particular cash flow is to be entered in)*
CashflowLoad(RunOpt) *(procedure that loads up cash flow, with different run options indicated by variable RunOpt)*

The parenthetical statements above would seem ideal for comments in the code itself, but they are mainly superfluous. The names are self-documenting in that respect.

In addition to naming things appropriately, make sure you have a consistent naming style so you don't create confusion. As just shown, I like the InterCapitalizationStyle of variable names, whereas others prefer underscores_between_words. It's a matter of taste. As well, many people like to use all caps to indicate constants, and might like to use different prefixes to indicate variable type; for example, "sFile" would be a string, "nFile" would be an integer and "dFile" would be a double-precision float. Having such prefixes helps when you're dealing with a language that is strict about variable type casting.

Finally, it is important to have a consistent style with regard to indentations for subblocks of code or line continuations, and think about putting spaces between "paragraphs" of code. The idea is to make readable code, where the structure is evident from the shape of indentation and spacing. The more readable you make your code, the fewer comments you will need to put in to help someone understand what it's doing.

3. Make it modular.
If you do the same sort of thing more than once, spin it off as its own procedure or function. This makes your code easier to read. Consequently, if a step in the procedure changes, you will only have to change it in one place as opposed to all the places where it occurred. This makes your code portable in that you will be able to call on the procedure more times with a single line of code, as opposed to copying, pasting and changing variable names in the appropriate places.

Here's a recent example from my own code. I had an Excel VBA macro that was supposed to load up asset cash flows from two separate scenario files. The original code looked like this (in very simplified pseudo-code form):

```
Open ScenarioFile1
Find asset CUSIPIdx
Find cash flow output columns for scenario 1
Pull out cash flows (loop)

Open ScenarioFile2
Find asset CUSIPIdx
Find cash flow output columns for scenario 2
```

Pull out cash flows (loop)
The code worked fine as just shown, but I modularized it so it was easier to read:

```
Call LoadSingleScen(ScenFile1, CUSIPIdx,
OutputColn1)

Call LoadSingleScen(ScenFile2, CUSIPIdx,
OutputColn2)
```

Later, I had to alter my runs so that I was comparing the output of four separate scenario files instead of just two. Boy, was I happy that I had modularized my code earlier. It was just a matter of adding a few more procedure calls and I was good to go.

In addition to setting off certain lines of code into their own procedures or functions, it's helpful to group procedures and functions together in their own files that can be popped on and off as needed. In VBA for Excel, you can have separate modules which you can export and import as needed. In C or C++ you can have linked files and libraries. I found this helpful in programming when I might have different models for partial withdrawals on a variable annuity with guarantees—I could remove the file with my baseline assumption function for withdrawal rates and link up the file with the new assumption. This makes it very easy to mix-and-match modeling structures.

4. Don't hardcode anything.
By hardcoding, I'm referring to having items such as parameters, file names, directory structures and specific Excel cell locations explicitly referenced in the body of the code. The only numbers that should appear in your code are 0 or 1 … maybe 2, but that's pushing it.

The reason for avoiding hardcoding is two-fold. In the case of Excel, it can be pretty obvious—the cell, row or column you want to reference may have moved. You get used to formula references in Excel updating when cells move around, but alas, the VBA code will not change with it. To get around this, I often use named ranges or read in column and row numbers from a worksheet where I've put a bunch of row() and column() references. References going obsolete due to spreadsheet change may seem limited to Excel, but it happens to C programs, too—input files can get moved to different directories or might get renamed.

The most common form of breaking this rule is hardcoding parameters for models into the code. Of course, it's highly unlikely that parameters will remain unchanged throughout all versions, so hardcoding parameters means you're going to have to sift through the code to find the numbers that need to change with model modification. You can't count on any parameter staying the same over time. Also, if the parameters that need changing appear more than once in the code, you have a chance for some really hideous results if only one line of code gets changed and the others are left untouched.

There are two ways around hardcoding: (1) have the relevant items defined as constants at the top of the code (or in its own constants module), or even better, (2) read them in as input items.

5. Have some sort of version control.
In this category, I've not been as good as I should be. I've started documents where I have kept track of my ever-evolving versions of spreadsheets and codes, but over time I don't keep up with it. When it comes to any type of documentation for a program, if the documentation is separate from the code itself, it's too easy to forget to update.

"You don't want to spend time coding if you already have perfectly good code from an old problem that can be copied for current use."

(continued on page 12)

So now I put version descriptions "up front." In C programs, I put it in comments in the section where I declare my constants. In Excel, I add an extra worksheet to note the code and worksheet changes I've made. Also, I make version notes cumulative. That way if I have to look for a previous version, the history is in my latest version and I can easily find it.

"… it is important to have a consistent style with regard to indentations for sub-blocks of code or line continuations. …"

It's good to keep previous versions because sometimes you need to replicate an earlier run, and using the earlier version of the code is easier than trying to hack your current code to its original form. Also, sometimes I have to scrap later work and go back to an earlier version, because product design has changed, or a particular business decision has been made. It's much better to be able to backtrack than to start from scratch.

Enjoy Programming

The results from cultivating these programming habits should pay off quickly, at least in terms of reducing the amount of time you spend replicating earlier programs or trying to interpret your own code.

I have included some Web sites that contain additional programming tips, though they come from more general programmers, and some of the tips may be language-specific. Most of them are coming from the perspective of programs that will be used by other people, but may still be helpful for those who are the sole producer and consumer of their own code.

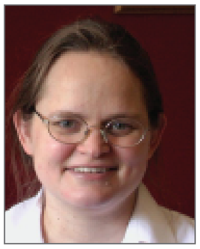
Moser, Kim. "Good Programming Practices: What to Do (or Not)"
http://www.kmoser.com/articles/Good_Programming_Practices.php

Moser, Kim. "Formatting Your Code: Why Style Matters"
http://www.kmoser.com/articles/Formatting_Your_Code.php

Chu, Philip. "Seven Habits of Highly Effective Programmers"
http://www.technicat.com/writing/programming.html

Zainvi, Syed Feroz. "15 Good Programming Habits"
http://ezinearticles.com/?15-Good-Programming-Habits&id=45825

So pass these tips on, incorporate them into your own work and you'll have living (instead of dead) programs. Enjoy!



Mary Pat Campbell, ASA, is a senior actuarial associate at TIAA-CREF. She can be reached at marycampbell@tiaa-cref.org.

Actuarial Value Ladder: Insurance Market Model

by Meg Weber

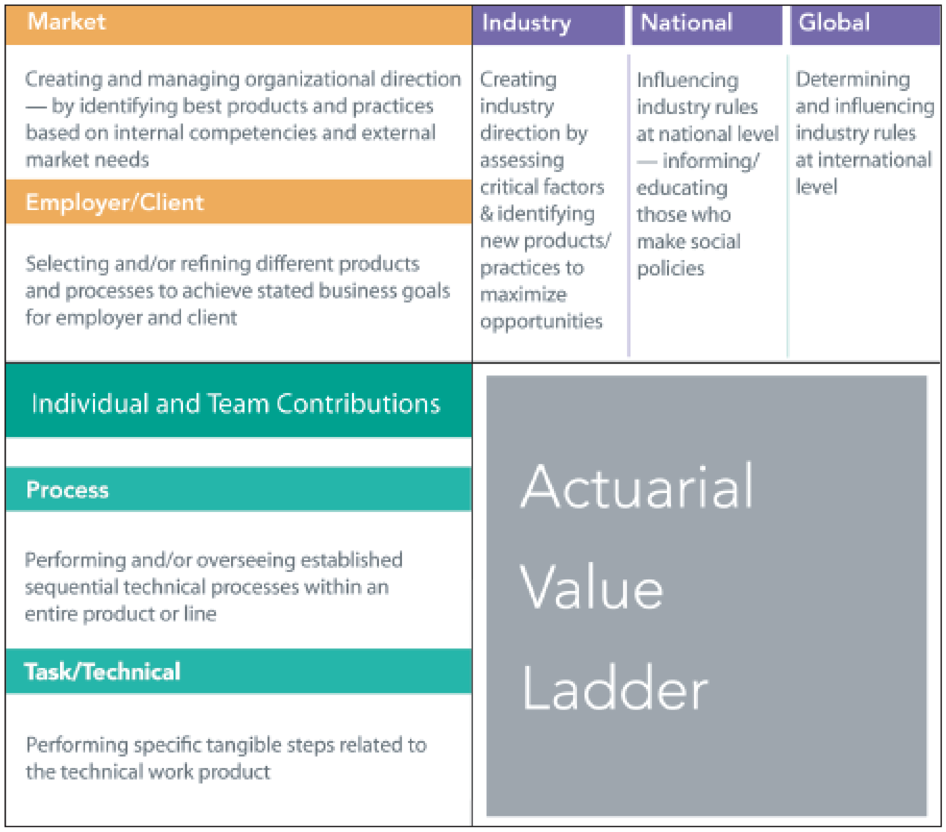
At the 2006 SOA Annual Meeting, the first prototype of the Actuarial Value Ladder was shared with the members of our profession. The concepts of this professional development tool are being incorporated in our 2007 Spring and Annual Meetings to assist attendees planning their sessions.

The Actuarial Value Ladder is a project of the SOA Marketplace Relevance Strategic Action Team (MRSAT) led by Chair Dan McCarthy. This career path tool articulates the value of the contribution actuaries can make and the competencies required across a full range of professional roles. As it matures, the Value Ladder becomes a powerful means to communicate the importance of the profession with employers. It also plays a vital role with regard to assisting actuaries in self assessment and developmental plans.

Each session at SOA meetings is aligned to a specific stage on the Value Ladder. As meeting attendees register, they can develop a "conference curriculum" that matches where they are in their careers and where they aspire to be at various points in their professional lives. The Value Ladder identification in meeting brochures, along with section sponsorship information, provides keys for registrants to ensure they get the most out of any event.

As indicated, the Value Ladder is a prototype. It is being used at these events as part of a series of "clinical trials." The MRSAT will be responding to feedback to improve the model. The development of more specific models to correspond to a wider range of actuarial careers is also planned.

"This career path tool articulates the value of the contribution actuaries can make and the competencies required across a full range of professional roles."


Market
Creating and managing organizational direction — by identifying best products and practices based on internal competencies and external market needs
Employer/Client
Selecting and/or refining different products and processes to achieve stated business goals for employer and client
Industry
Creating industry direction by assessing critical factors & identifying new products/ practices to maximize opportunities
National
Influencing industry rules at national level — informing/ educating those who make social policies
Global
Determining and influencing industry rules at international level
Individual and Team Contributions
Process
Performing and/or overseeing established sequential technical processes within an entire product or line
Task/Technical
Performing specific tangible steps related to the technical work product
Actuarial Value Ladder

Meg Weber is director of section services at the Society of Actuaries in Schaumburg, IL. She can be reached at mweber@soa.org.

Moving Ideas Forward

ANNUAL MEETING & EXHIBIT

SOA07

October 14-17, 2007
Marriott Wardman Park
Washington, D.C.

Mark your calendar and plan to participate in the SOA'07 Annual Meeting & Exhibit. We'll be offering insights into how to move your ideas off paper and into action. We're also planning more great seminars and networking opportunities—and of course, a few surprises.

WE'VE ALREADY LINED UP THESE EXCITING SESSIONS:

BUSINESS INTELLIGENCE FOR ACTUARIES: TOOLS, TECHNOLOGIES AND APPLICATIONS

Monday, October 15 10:30 a.m. – Noon

Business intelligence is a mainstream technology solution to the business decision-making process. This session will cover how this solution set is being adopted in the actuarial environment. We will discuss a broad range of topics including foundational data repositories to the data presentation tools and applications that actuaries will use to improve their analytical capabilities, financial reporting and decision-making processes.

DATA QUALITY: PLAYING WITH MATCHES

Tuesday, October 16 2:30 – 4:00 p.m.

This presentation is a primer on editing, imputation and record linkage for analysts who are responsible for the quality of large databases, including those sometimes referred to as data warehouses. Practical help regarding how to make informed and cost-effective judgments about how and when to take steps to safeguard or improve the quality of the data will also be shared. This session should be of particular interest to anyone whose day-to-day professional life involves data quality issues.

We look forward to seeing you there!

More information about SOA'07 Annual Meeting & Exhibit will be available in the near future at *www.soa.org*. Registration opens the end of June 2007.

Actuaries
Risk is Opportunity.℠