

CompAct

TECHNOLOGY SECTION

"A KNOWLEDGE COMMUNITY FOR THE SOCIETY OF ACTUARIES"

● Inside

Articles Needed for the *CompAct* Electronic Newsletter _____ 2

Goodbye to All That _____ 3
by *Phil Gold*

Letter from the Chair _____ 4
by *Paula Hodges*

An Analytics Manifesto _____ 5
by *Neil Raden*

Open and Closed Code Myths _____ 12
by *Kevin Pledge*

Open Versus Closed Software Code _____ 14
by *Mark Evans*

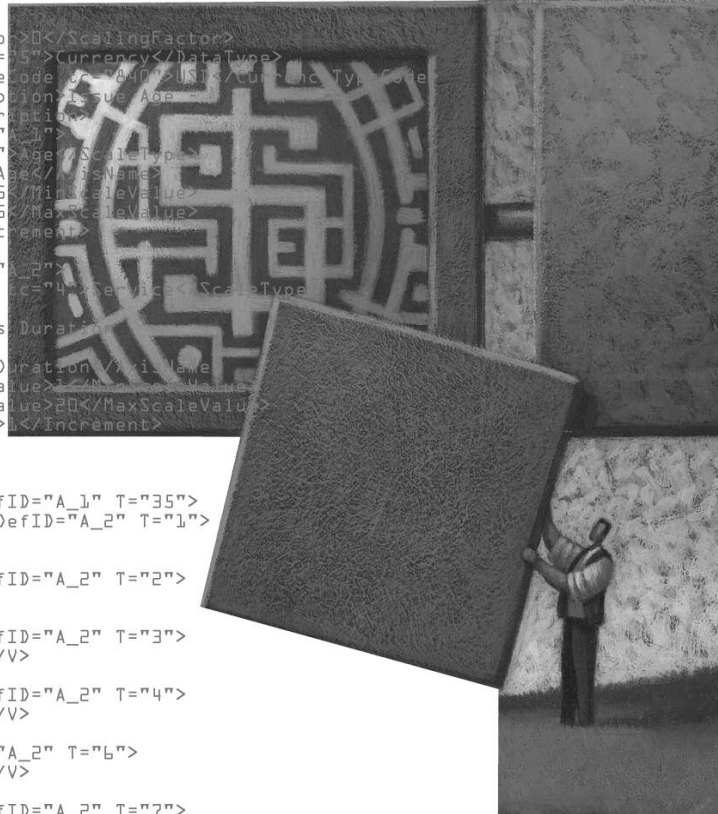
Open or Closed Code: A Response to Mark's Article _____ 16
by *Nazir Valani*

Annuity Calculations Using SQL _____ 18
by *Sheila Silva*

Actuaries

The Best-Kept Secret in Business™

```
<?xml version="1.0" encoding="UTF-8" ?>
- <XTbML id="Table1" Version="XTbML2.9.91"
  xmlns="http://ACORD.org/Standards/Life/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ACORD.org/Standards/Life/2
  XtbML2.9.91.xsd">
- <ContentClassification>
  <ContentType tc="45">Cash Value</ContentType>
  <TableName>Tabular Cash Value</TableName>
</ContentClassification>
- <Table>
  - <MetaData>
    - <ScalingFactor>0</ScalingFactor>
    <DataType tc="5">Currency</DataType>
    <CurrencyType code="USD" Unit="Currency Code"
    <TableDescription>Issue Age
  Duration</TableDescription>
    - <AxisDef id="A_1">
    <ScaleType tc="3">Age</ScaleType>
    <AxisName>Issue Age</AxisName>
    <MinScaleValue>35</MinScaleValue>
    <MaxScaleValue>45</MaxScaleValue>
    <Increment>1</Increment>
    </AxisDef>
    - <AxisDef id="A_2">
    <ScaleType tc="4">Services</ScaleType>
    - <!--
      also known as Duration
    -->
    <AxisName>Duration</AxisName>
    <MinScaleValue>1</MinScaleValue>
    <MaxScaleValue>20</MaxScaleValue>
    <Increment>1</Increment>
    </AxisDef>
  </MetaData>
  - <Values>
    - <Axis AxisDefID="A_1" T="35">
      - <Axis AxisDefID="A_2" T="1">
        <V>0</V>
      </Axis>
    - <Axis AxisDefID="A_2" T="2">
      <V>0.37</V>
    </Axis>
    - <Axis AxisDefID="A_2" T="3">
      <V>14.81</V>
    </Axis>
    - <Axis AxisDefID="A_2" T="4">
      <V>29.81</V>
    </Axis>
    <Axis AxisDefID="A_2" T="6">
      <V>61.56</V>
    </Axis>
    - <Axis AxisDefID="A_2" T="7">
```



Technology Section Newsletter
Issue Number 21
October 2006

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

Nariankadu D. Shyamalkumar

CompAct Editor
Assistant Professor
Statistics and Actuarial Science
241 Schaeffer Hall
The University of Iowa
Iowa City, IA 52242-1409
phone: 319.335.1980
fax: 319.335.3017
e-mail: shyamal-kumar@uiowa.edu

Technology Section Council

Philip Gold, Chairperson
Paula M. Hodges, Vice-Chairperson
Timothy Lee Rozar, Treasurer/Secretary
Joseph Alaimo, Council Member
Charles S. Fuhrer, Council Member
(2006 Spring Mtg. Prog. Comm. Coordinator)
Kevin J. Pledge, Council Member
(2006 Annual Mtg. Coordinator)
N.D. Shyamalkumar, Council Member
Dean K. Slyter, Council Member
Stephen J. Strommen, Council Member

SOA Staff Contacts

Joe Adduci, DTP Coordinator
jadduci@soa.org

Meg Weber, Director, Section Services
mweber@soa.org

Susan Martz, Project Support Specialist
smartz@soa.org

Glenda Maki, Senior Editor
gmaki@soa.org

Facts and opinions contained in these pages
are the responsibility of the persons who
express them and should not be attributed
to the Society of Actuaries, its committees,
the Technology Section or the employers of
the authors. Errors in fact, if brought to our
attention, will be promptly corrected.

Copyright© 2006 Society of Actuaries.
All rights reserved.
Printed in the United States of America.

Articles Needed for the *CompAct Electronic Newsletter*

Your help and participation is needed and welcomed. All articles will include a byline to give you full credit for your effort. *CompAct* is pleased to publish articles in a second language if a translation is provided by the author. For those of you interested in working on *CompAct*, several associate editors are needed to handle various specialty areas such as meetings, seminars, symposia, continuing education meetings, new research and studies by SOA committees and so on. If you would like to submit an article or be an associate editor, please call Nariankadu Shyamalkumar, editor, at 319.335.1980.

CompAct is published as follows:

| Publication Date | Submission Deadline |
|-------------------------|----------------------------|
| January 1, 2007 | October 15, 2006 |
| April 1, 2007 | January 15, 2006 |
| July 1, 2007 | April 15, 2007 |

Preferred Format

In order to efficiently handle articles, please use the following format when submitting material:

Please e-mail your articles as attachments in either MS Word (.doc) or Simple Text (.txt) files. We are able to convert most PC-compatible software packages. Headlines are typed upper and lower case. Please use a 10-point Times New Roman font for the body text. Carriage returns are put in only at the end of paragraphs. The right-hand margin is not justified.

If you must submit articles in another manner, please call Joe Adduci, 847.706.3548 at the Society of Actuaries for assistance.

Please send electronic copies of the articles to:

N.D. Shyamalkumar

Technology Section Editor
e-mail: shyamal-kumar@uiowa.edu

Thank you for your help.

Goodbye to All That

by Phil Gold

It seems like just yesterday that I started my three-year stint on Council. I didn't know what to expect at the time but my parting advice is, "The water's warm—jump in!"

During these last three years I have met many people, made lots of new friendships and cemented some old ones: I have been exposed to the workings of the SOA to a degree I did not expect. You may not know this but as Section Chair and Vice-Chair, you get invited to regular meetings in Chicago as well as monthly Internet based sessions. You learn all about consent agendas and other techniques that can find good application not just here but also back in the office. By attending and speaking at meetings you get to know your own subject much better, you meet others with different perspectives, and you quickly find out just how heterogeneous a group we actuaries are.

Is there a lot of work? Sure. Is it worth it? Certainly. Is the SOA changing? You bet!

The title of my piece, "Goodbye To All That," is a little tongue in cheek. You don't get rid of me that easily. First of all, I'm still King, sorry, Chair, until Paula secures my abdication at the Annual Meeting in Chicago. Second, when you step down, the next Chair may invite you back to sit among the "Friends" of the Section Council. Finally, the SOA has your number and there's no telling how often they may call.

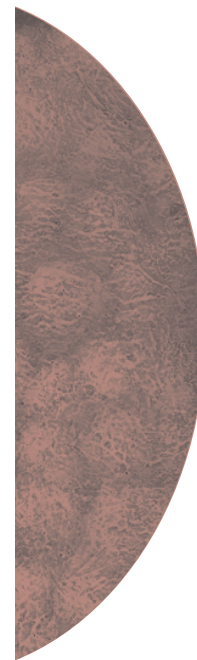
To be honest, I have been able to float through this year's responsibilities because I am blessed with a superb set of Council members who have done all the heavy lifting. One huge thank-you to all the members and friends who have done so much so well this year. Guiding us and facilitating our efforts is a very strong and friendly team at the SOA HQ, and, if I may, I will single out Meg Weber and Sue Martz for their support.

At the beginning of the year I promised to improve the communications the Section has with its members. Since then, we've had four newsletters, bimonthly Tech Updates, an updated Web site, a new initiative on Standard Scenario Formats and a superb program and a big social event should be awaiting us in Chicago.

I can promise you one last thing. I will leave the Section in most capable hands. 🖥️

Cheers, bye, shalom, adieu.

Phil Gold
Technology Section Chair 2005-6



Phil Gold, ASA, FIA, MAAA, is a founding partner of GGY and chairperson of the Technology Section. He can be reached at pg@ggy.com.

Letter from the Chair

by Paula Hodges

As the Technology Council begins a new year, it seems fitting to reflect on the changes that have shaped our industry in recent decades. I recently completed reading "The World is Flat" by Thomas L. Friedman. The theme in the book emphasized to me how much our lives have been changed dramatically and powerfully, due to forces that are moving at the speed of light around us. We need to continue to step up to the challenge of utilizing the capabilities around us, because if we don't do it, someone else will.

To illustrate these changes in our profession, I recall stories from one of my mentors, from early in my career. He used to tell of the type of work assigned to him when he was an actuarial student in the early 1960s. At that time, actuarial students were assigned to such mundane tasks as manual calculation of dividend tables and interpolating extended term insurance. This was all done with paper and pencil, of course, and assigned to the actuarial students, as they were the resources capable of the math and attention to detail required for these tasks. Also during that era, mortality tables were typed out and kept in file cabinets. Ratebooks were sent to the printshop for typesetting, and then sent to the agents to help them calculate premiums for their clients.


Flash forward to 2006: Agents don't use ratebooks anymore—they are connected to home office and have sophisticated illustration software that not only quotes the premium that their client will pay, but can also produce complicated financial planning scenarios, using flexible premium products. And they get all this with just a few clicks of a mouse. Actuarial students, no longer doing "grunt work," are running complex modeling software that has the capability of executing hundreds, if not thousands, of financial scenarios.

This metamorphosis of our work and our industry was only possible by the increased

technology capabilities that are available to us. It is important that we, as actuaries, foster a great respect, if not a keen interest in technology developments. It is this technology that will channel our key competencies into designing our products, pricing our assumptions, analyzing capital positions, stratifying market segments, and dissecting the details of our own companies' financial statements.

Not so many years ago, it served us well when we were spreadsheet wizards. In today's world, we need to have technology/actuarial specialists who understand business intelligence reporting protocols, code optimization, and XML standards. By building a forum for collaboration and networking, this group of technology/actuarial specialists can contribute considerably to the acceleration of innovation in the various actuarial disciplines.

It is in this spirit that the Technology Section was born in 1992 as the Computer Science Section. It is this same spirit that Phil Gold, our chair for 2005-2006, brought forth the power of the volunteers of the section. He reenergized the *CompAct* newsletter, kept us updated with his Tech-Update e-mails, and encouraged our R&D by starting up a working group on standardization of scenario generation. Thank you Phil, for your work and leadership over this last year.

I look forward to serving as Chair of the Technology Council for 2006-2007. If you are a member of the section, I urge you to read our mission statement on the SOA Web site. Contact me, or any member of the Council, to contribute to our newsletter, help with the Technology sessions at one of the SOA meetings, or if you have any other contributions toward our goals. 

Paula Hodges
Chair, Technology Section 2006-2007



Paula M. Hodges, FSA, MAAA, is manager of Modeling Strategy with Allstate Financial in Lincoln, NE. She can be reached at phodg@allstate.com.

An Analytics Manifesto

by Neil Raden

The size of the Data Warehousing/Business Intelligence industry pales in comparison to the operational software business. In 2005, total spending for software (globally) reached over \$100 billion¹ according to IDC, but BI's share was less than 5 percent. The stated purpose of enterprise systems is to improve organizations' effectiveness through better utilization of information, especially for decision-making. Yet somehow, analytics have taken a back seat role in corporate IT more than 40 years. The forces of technology and business are about to change that.

The rapid acceptance of open standards like Web Services, which makes Service-Oriented Architecture possible, provides an appealing opportunity for re-architecting business process software as a set of cooperative services. The promise is increased alignment through vastly improved agility, reduced maintenance costs, which account for up to 75 percent of IT budgets overall,² and a reduction in integration expenses, which make up a large portion of IT development budgets.

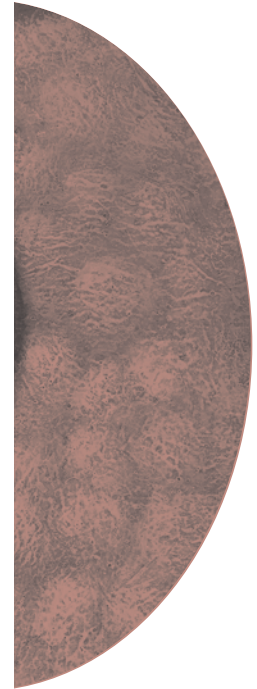
The element missing from most road maps, however, is business analytics. Agile organizations need analytical insight embedded into their operational processes to "close the loop" between analysis and action. The lag between actions taken, results tracked and conclusions drawn has been getting increasingly smaller, but the last step—connecting analysis to operations in a continuous cycle—is restricted by existing tools, methods and approaches.

Many current analytical tools limit and simplify the models and scale of data that can be investigated because of their client-based architecture. Even in server-based Business Intelligence (BI), many of the tools scale poorly and do not leverage the data resources already available. The loosely-coupled³ nature of SOA will expose these shortcomings and favor those vendors that can provide optimal

functionality for the new architecture. Current tools for analytics are hampered by a lack of common semantics, too many proprietary engines, layers of stand-alone functionality and poor data federation capabilities.

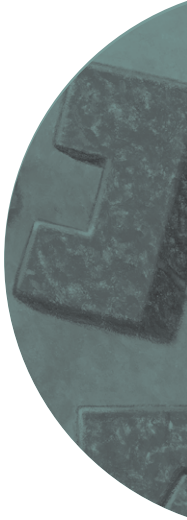
A combination of forces is working to tip the scales in favor of a new generation of analytics:

- **Moore's Law:** Analytical schemes today are still managing from scarcity. The continued rapid increase in capacity and decrease in relative cost of hardware and bandwidth allows for the faster and broader analysis of data.
- **Service-Oriented Architecture:** Open standards foster an environment where analytics can cooperate with operational processes without disrupting or interfering with the smooth operation of these systems and processes.
- **Maturity:** Most organizations, and some of the people in them, have some experience with analytics now and are open to improvement.
- **Business Process Commoditization:** Visionaries like Davenport⁴ are getting the word out that a lot of the work in process optimization has been done.
- **On Demand:** Opening business up with the Internet creates an entirely new timetable for conducting business—on demand.
- **Semantics:** The rapidly expanding field of Semantic Technology will power analytics to new levels, fueled by open standards for ontologies such as RDF and OWL. These are XML variants for representing ontologies as 'triples'—subject, predicate, object, which can be classified (in binary form) as graphs.



Neil Raden is president & CEO of Hired Brains, Inc. in Santa Barbara, CA. He can be reached at nraden@hiredbrains.com.

(continued on page 6)



- Hegemony of a few vendors and influence of analysts: These can act as negative forces, but once enough momentum is gathered, they tip abruptly.
- The status quo is simply not good enough. BI has not reached as many people as it should, and has not been as effective as it could be.
- Vision of companies like SAP that understand that analytics is an integral part of almost every application.
- Analysis in isolation is not effective; it must close the loop within business action.
- Commodity business processes will be outsourced, increasing the need for reliable measurement.
- The requirements for real-time will grow as the availability of reliable real-time tools increases.
- The effects of Moore's Law will gradually move thinking about analytics from "managing from scarcity" to capitalizing on its capabilities.

The fundamental nature of data warehousing and Business Intelligence today is the bulk gathering of data and generic presentation of it to a wide audience, predominantly for reporting and extraction to spreadsheets. Analytics must be a central part of business processes and the tools, techniques, products and best practices are just now being formulated.

Manifesto: New Rules

The entire field of analytics has been, to date, driven by the marketing of BI vendors and their influential customers. As analytics emerges from its various niches and proves its centrality to operational and strategic roles, a clear set of rules is crucial:

- To achieve its mission, the use of analytics at all levels of skill and function, has to be pervasive in organizations.
- Analytics, even complicated analytics, does not have to be difficult and should not be the preserve of a cadre of statistical experts.
- Analytics is useful at achieving ROI at the operating level. At the same time, it plays an important role in setting and managing strategy with tools like CPM (Corporate Performance Management). Going forward, these roles must be aligned, consistent and seamless.
- Analytics cannot just inform; it has to be active, stitched into the fabric of work.
- Embedded analytics needs to be composed of standard services with consistent function and metadata across the enterprise.
- Service-Oriented Architecture will move analytics from a departmental to an enterprise pursuit.
- Semantic technology promises to be the fuel behind increased use, utility and value in analytics. The application of industry standards to the building, discovery and merging of ontologies will vastly improve everyone's ability to inform themselves of the meanings of complex terms and their relationships.
- Metadata must rise above proprietary standards; abstraction layers must be the preferred path for access to data.
- Analytical tools will eventually learn how people work, not vice-versa. In the meantime, it will still require customization and configuring to produce analytical tools that are relevant and understandable to a wide audience.
- In an SOA world, BI tools that bundle query, interface, output, metadata and calculate will have to gradually un-bundle their services.
- What makes BI difficult for people is the lack of understanding of the data, models and tools. Analytics has to be relevant to work that people do and make their work easier, not more difficult.

A new generation of analytical tools is needed that can be seamlessly embedded in business processes. Ideally, they must:

- Extend from viewing of results to interactive exploration of data and models;

- Be as tightly connected to the strategy of the organization as it is to the minute operations;
- Foster collaboration and group decision-making;
- Provide the ability to share knowledge models and techniques across the enterprise and beyond, through the easy assembly of components by business stakeholders.

Today, fewer than 20 percent of knowledge workers in organizations avail themselves of analytics.

Factors Affecting Analytics

Before e-business and the Internet, before business process reengineering, organizations were able to function in a timeframe that seems inordinately long by today's standards. Few decisions needed to be made immediately, no one was in danger of losing their market to a competitor without a protracted fight, and the process for reviewing results and deciding on tactics was measured in weeks, not hours or minutes. It was in this environment that our current models for analytics and business intelligence were formed. It explains why data warehouses are still a largely batch process, why business intelligence software is still focused on "seats" or individuals instead of communities of work and whole enterprises. But today, the luxury of latency is gone. On demand is not just a clever TV commercial; it is the way business is being conducted.

Business Intelligence is still a departmental or even individual affair in most companies. The move to Service-Oriented Architecture will force IT managers to look at analytics as an enterprise asset. Because analytics will become an integral part of many operational systems, the current situation, with many different BI tools in place and skills spread too thinly to be useful, will become unwieldy. Standardizing on those analytical services that are best suited to the propagating "composite" applications (operational and analytical), instead of each department's selection,

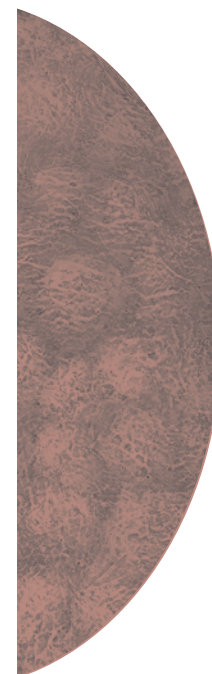
will have the effect of elevating the visibility of analytics. It does not mean that all organizations will need to apply advanced analytics to their business processes; they will need to have faith in the output of analytical processes. There are various ways for this to happen, but one method that is gaining notice is to create a team of specialists who perform most of the advanced analytics on a centralized basis, and who have the confidence of the senior executives of the firm. Their work, finding the underlying causes and relationships and predictors, can be reduced to models with a set of parameters that can be run repeatedly. FICO[®] credit scores are a good example of this, where Fair Isaac develops and continuously tunes a model that can be rendered as a series of variables that produces a credit score.

This premise, most recently advanced by Tom Davenport,⁵ historically has some drawbacks, but they were likely caused by organizational separation of the modelers and those who used the models. Nevertheless, Davenport and others writing on this subject do at least highlight the need for these capabilities and focus on executive acceptance of analytics, which is crucial. In an SOA world, it is likely that advanced analytics will be palatable to a wide audience because it will be transparent.

Semantics

A major impediment to the use of BI tools is not the mastering of the tools themselves. This is caused by a lack of understanding of the data, what it means, and its relevance to the task at hand.⁶ Metadata management is positioned as the solution to the problem, and it certainly is, but the current discipline of metadata management is not a complete answer because, in particular, existing BI tools provide incomplete and proprietary metadata. In Figure 1, the location of different sources of metadata can be spotted in this *simplified and idealized* chart of information flow for analytics. A single BI tool may actually have as many as five different metadata structures, but gathering them into a single metadata

(continued on page 8)



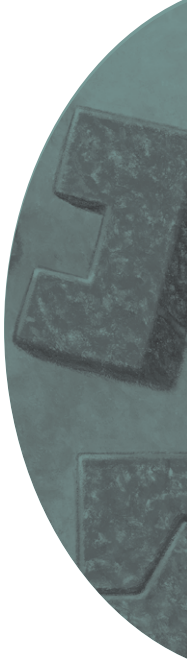
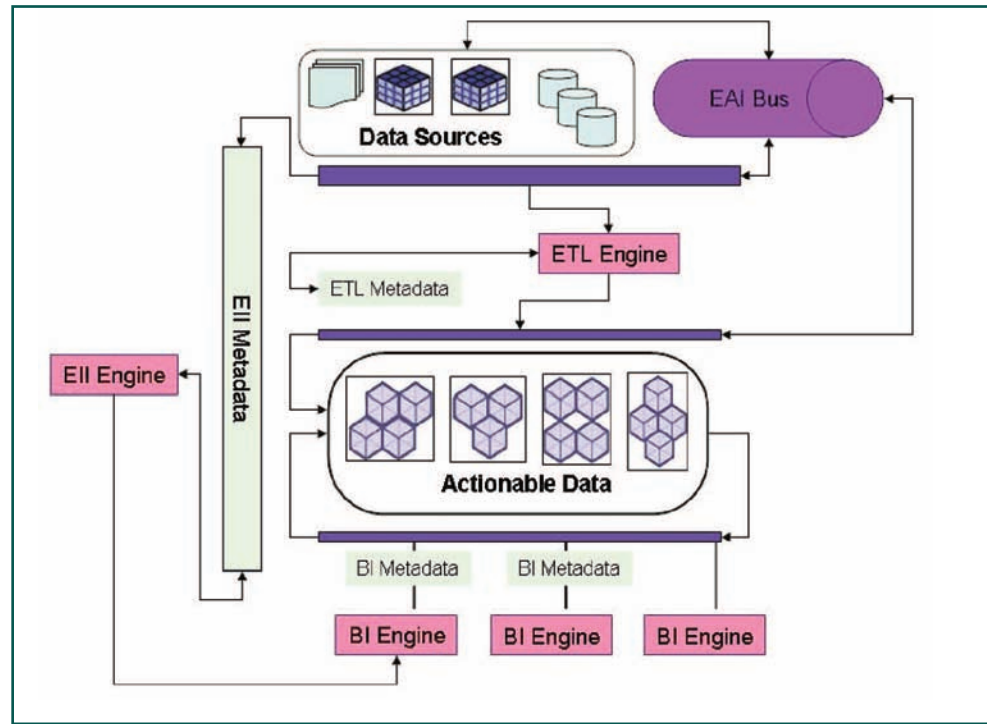


Figure 1: Fractured Metadata in Current BI Architecture



repository without altering their form or function does not solve the problem.

Ideally, metadata should allow for different tools used in different locations (logical or physical) to be able to exchange information about their semantics and operation. The lack of uniform metadata across applications makes it difficult to collaborate and hinders standardization. BI metadata comes in roughly three flavors:

- **Production** – Describes the movement and translation from one data source to another, establishes ownership and logs updates.
- **Catalog** – Definition of tables and attributes.
- **Presentation** – Additional data and calculation definitions, report layouts, preferences and roles in the reporting/analysis tool.

A great deal of work has gone into defining

metadata standards and attempting to open up the metadata, but to a lesser degree, competitive pressures between the vendors impede the effort. To a much greater degree, however, technology itself stands in the way of a useful metadata solution. Using relational databases and relational modeling techniques as a solution may be inadequate to capture the richness and nuance of the data, models and conditions in even the simplest business. Using extended relational modeling, such as UML, offers some marginal improvement, but all of these schemes effectively leave metadata in a passive state.

Semantic technology holds the promise of breaking through the metadata ice jam. Semantic technology is a broad field, but the rapidly growing commercialized part of it is called *ontology*. An ontology goes beyond definition to capture the semantic meaning of things and, as a result, creates a structure from which a machine can draw inference. In a metadata query, all of the knowledge need-

ed to frame a question is in the query, composed by the query writer. In ontology, the relationships that are captured are capable of revealing, through a process of introspection and inference, more information than was consciously placed into it. In addition, ontologies are constructed in a language that is based on XML, the Resource Description Framework (RDF), or built with the Web Ontology Language (WOL), all of which are open standards that are managed by the same committee, the W3C that is responsible for Web Services.

Semantics will play a role in the orchestration of Service-Oriented Architecture in a multitude of ways. The discovery of services using UDDI and WSDL is very limited, but semantic extensions to these services make it possible to find a service with a conceptual search such as “Find a seasonal smoothing function that is used for consumer products.” The massive amounts of data in data warehouses no longer need to be limited to a single, isolated definition. Semantics can point out which elements are dependent on others or are predictors. Semantics can be extremely valuable even when they are incomplete. Semantics can be constructed incrementally by many people simultaneously and incorrect entries can be handled by the semantic engines. There is even a name for ontologies that are built by people over time—*folksonomies*.

Semantic technology promises to supercharge analytics by removing the major obstacle to using analytics by a broad cross-section of the population—lack of understanding. In the near future, metadata built with semantic technology will glue all of the disparate services together, with meaning replacing definitions and patterns. Current metadata approaches are largely based on data; with semantics, metadata can marry data, process, application, use, security—virtually everything. Furthermore, it will be able to exchange this information freely because it is based on open standards that conform perfectly with an Service-Oriented Architecture approach.

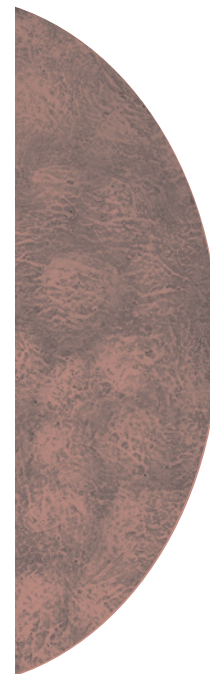
Principles

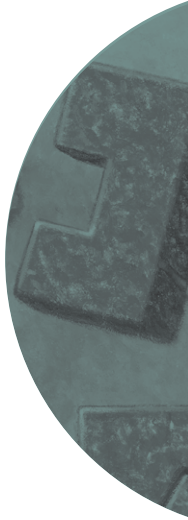
Many current analytical tools limit and simplify the models and scale of data that can be investigated because of their client-based architecture. Even in server-based BI, many of the tools offer architectures that scale poorly and simply cannot leverage the volume and richness of the available data. To claim that a product is scalable because it can handle dozens of servers, but only 10-12 users per server, and perform no real load balancing or work distribution across servers, only complicates the result. IT groups are forced to limit access to data warehouses by concealing detailed data, implementing query governors and/or limiting live queries to certain times of the day. As a result, despite huge data warehouses housing the freshest and most detailed data available, the models that are used in BI are typically summarized, simplified and denatured to fit into the limited computing capabilities of most BI tools.

What next-generation analytics promises, particularly as a result of loose coupling and open standards, is the horsepower to finally exercise the investment in data warehouses. However, these limitations are not the only factors that limit the reach of analytics today—they may not even be the most important factors—but they are the ones that can be dealt with easily through technology. Changing the way organizations use information (analytics in particular) to run their business, breaking down the practices that separate knowledge workers from useful tools and information, and educating people that passive receipt of information is not sufficient and levels of self-service and investigation are not beyond anyone’s intellectual reach, all are challenging problems to solve.

Manipulating very complex models and walking through reasoning, layers of detail and concepts does not have to be difficult. Quality analytical tools should adjust to the user’s level of skill, and more skilled users will be able to create analyses for others to share.

(continued on page 10)





And finally, because all aspects of the analytical workplace, including data, relationships, models, assumptions and report objects will be recorded in a semantics-oriented abstraction layer. This will be similar to, but much more robust than, today's metadata. The sharing of knowledge, the ability to use encoded knowledge in new and creative ways, and the emergence of collaborative analytical environments will become the "killer apps" of the Serve-Oriented Architecture world.

Analytical Functionality

In summary, here is a partial list of analytical functionality that is needed in the next generation of Serve-Oriented Architecture-enabled analytical tools. Many of these features already exist today, but in the table below, the column to the right describes how they need to change.


| Analytical Functionality | |
|--|--|
| Current State | Desired Future State |
| OLAP | Separation of navigation/presentation from engine; OLAP-like manipulations, such as drilldown, need to be available as services without human interaction. |
| Descriptive statistics | Common statistical engine across all applications boosts understanding. |
| Threshold-based alerting | Event-based service watches and alerts and is configured for a multitude of processes; this feature should be generalized and not linked to a particular data source or schema. |
| Predictive modeling | Instead of a variety of specialized tools implemented by disjointed groups, predictive modeling tools can be implemented as shared and reusable services for all of those qualified to use them and parameterized models derived from their discoveries implemented as services and embedded in many operations. |
| Optimization | A generic optimization engine with different add-on optimization engines, with domain intelligence, applied to the right problems. |
| Automated decisioning | Combination of event manager, predictive modeling, optimization, rules engines and other decision tools to take over the task of low-risk decisions and inform those responsible for higher risk ones. |
| Dashboards that learn what you want and how you want to see it | Dashboard software, abstracted from most of its current functionality, focused on learning patterns and needs of clients. |
| Ontologies in place of proprietary rules engines | Semantic technology in open standards such as RDF using conceptual search techniques to draw inferences without explicit rules. |

Conclusion

In conclusion, analytics are crucial for both making policy and managing it. In the past, different vendors supported different groups within organizations to provide tools for planning, budgeting, forecasting, statutory reporting and other high-level, policy-making and strategic requirements, on the one hand, and analytical and reporting functions after the fact on the other. These efforts were disjointed, incompatible and often conflicting. To the extent that operational systems employed analytics, they were typically embedded in the software and neither reusable nor transparent to the rest of the operations.

Thanks to the onward march of technology, and to Moore's Law and Service-Oriented Architecture in particular, the gap between strategy, operations and analytics is closing. The Service-Oriented Architecture environment provides an infrastructure to support dynamic reuse, which requires new classification systems for BI services and policies that are equivalent to operational service policies. If the function of operational and analytical software is going to blend, then these elements will have to come into compliance too. Analytical software as it is today is not entirely ready to be part of a distributed, enterprise computing architecture. Many of the products, including the market leaders, have a lot of work to do before their products can perform effectively as a set of loosely coupled asynchronous services.

There is still a great deal of work to be done in analytics. The past 10 years were focused more carefully on data management than on the application of analytics. Though the problems of data quality, data freshness and data conformity have not been solved, and issues about metadata and master/reference data are still brewing, an enormous amount of progress has been made. Ten years ago, data warehousing was a good idea, but implementing one successfully was very difficult. All of the time and attention spent lessening that risk came at the expense of understanding good analytical technique and architecture. Today, there is no time to waste. Because analytics are critical at both

the strategic and operational levels, engineering analytics to perform smoothly across the organization is a top priority. Every organization will need to find their own unique way of implementing it, but we will all depend on the vendors to provide the tools and the technology to get it done. 

Endnotes

¹ Enterprise Systems, "IT Spending Rebounds," February 15, 2005, <http://www.esj.com/news/article.aspx?EditorialsID=1280>.

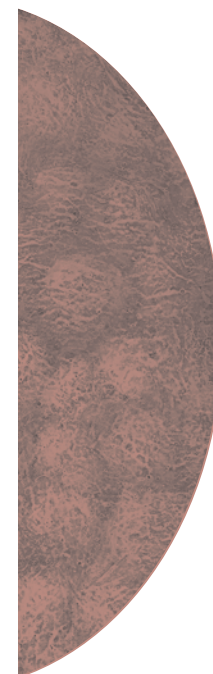
² "AVM Tools Will Reach \$500 Million to \$700 Million By 2008," July 22, 2005, Forrester.

³ Loosely coupled refers to the ability of services to be anywhere on a network, without regard to platform or development technology provided they can be discovered and invoked through the standard methods.

⁴ Thomas H. Davenport, The Coming Commoditization of Processes, *The Harvard Business Review*, June, 2005.

⁵ Thomas H. Davenport, Competing on Analytics, *The Harvard Business Review*, January, 2006.

⁶ <http://www.intelligententerprise.com/show Article.jhtml?articleID=19502113>.



Open and Closed Code Myths

by Kevin Pledge

Badly designed systems, a curiosity about the underlying code and a “not-invented-here” stubbornness result in the proliferation of myths that support open code. However, anyone contemplating the use of open code systems needs to give serious consideration to the reality behind these myths. This article discusses some of the myths and misconceptions associated with open and closed code.

Before discussing these issues it is necessary to start with some definitions:

- Closed code may be more appropriately referred to as “vendor maintained;” such a system will include documentation and will provide users with a method to validate results. Some closed code systems may also include “hooks” for user code or user defined reports.
- Open code is a framework that enables a large number of developers to contribute to the development. Users can review and modify the source code.

This article does not set out to discuss whether vendors should expose their source code; such an action would likely be commercially unviable and does not mean that the code can be modified.

Myth #1: Open code, or reviewable code, is easier to validate.

Busted: Reviewing code is difficult and time consuming. For any complex system, reviewing code is not a practical way to understand the functionality of that system. In fact, trawling through code is more useful for learning about the structure of the code than the validity of the application. A well designed system will include descriptions of

the methodology employed and output to enable testing.

Furthermore with modified open code, since no other companies are using the same code, there is less peer review.

Myth #2: Open code can be customized and extended.

Confirmed (but it isn't simple): Modifying a system is more than modifying code; it requires an understanding of the architecture. A change made in one place may require changes in numerous other places in the code that the user may not be aware of.

If multiple people in a single organization make changes, you either end up with numerous incompatible versions of the software, or you throw all the changes into one version. In that case, the individual changes may interfere with each other, resulting in anything from catastrophic failure to subtle difficulty in detecting errors.

The intention of open code is to provide a framework for changes to become part of the core program. In practice it is really difficult to develop a good framework without applying it to a specific problem. If the customizations cannot be incorporated into core code, this will result in splinter versions of the system as has occurred with Unix, or client specific versions that the client is required to maintain.

Myth #3: Allowing open modification results in better performance.

Busted: The idea that a user base would improve the code is naive. While there may be some attempts to improve the code, the potential for improvement is restricted by the framework. In fact, the vendor would also be constrained from making performance improvement, particularly if these involved changing the programming language, using



Kevin Pledge, FSA, FIA, is president and CEO of Insight Decision Solutions in Markham, Ontario. He can be reached at kpledge@insightdecision.com.

proprietary routines from a third party, or even using code that would be more difficult to interpret. The result is a system constrained by its original designed parameters, likely to be replaced by applications written in newer generation languages or built on newer platforms.

Myth #4: Open code contains less bugs.

Busted: The assumption here is that users will contribute to fixing bugs. The reality is that no one actually wants to fix other people's bugs and incorporating the solution may conflict with other development. The developer mindset is one that likes to solve problems for themselves and develop reusable code, rather than reuse code. It is more important to ensure that the vendor will respond to bugs and issues in a timely manner.

Myth #5: Open code enables self-support.

Busted: Supporting a system requires coding expertise and detailed knowledge of the system design, including features that may have been incorporated for future enhancements. Also custom modifications make supporting the software, including providing upgrades, difficult for the vendor or a third-party.

Myth #6: Open code wins on cost.

Busted: Although there are fully functional versions of open code applications that do cost nothing, these generally are not commercially supported and may not have adequate documentation or technical support. To develop, deploy and maintain open code software involves many costs, just like any other software.

Myth #7: Open code is license free.

Busted: Open code is still covered by a license agreement. In fact, the license may be more restrictive than you expect, such as clauses that can change the terms of the license with 30 days notice. Furthermore, you should understand who is liable if a contributing

developer has used proprietary code and the possible consequences of this.

Myth #8: Open source code best allows the incorporation of contributions from a wide range of users.

Busted: While user (and developer) feedback is the driver behind open source software, it needs to be managed and coordinated. It is difficult to incorporate unplanned input and multiple changes made at the same time may well conflict with each other. Furthermore, closed code systems may license components from other vendors, for example, .NET components can be licensed to provide non-core functionality.

Conclusion

These aren't myths because they're never true, and in many cases closed code does not automatically provide a better solution. While there may be examples of successful use of open code software, these are generally limited to smaller applications or are successful for a short period of time in situations where one developer is available to dedicate its time to support the application. When that developer leaves the company it is difficult to continue support for the application and going back to the original will likely lose much of the client customization.

Vendors do not only protect source code for their benefit; it also protects the client from the problems that can occur with unmanaged development. Vendors, with client input, plan the direction of product development and employ sophisticated processes to manage source code and ensure the legal integrity of the code. It is important that they recognize the need for the client to understand the application, be able to validate the processes employed and listen to the client with regard to the direction product development takes. Whether the first two points are best achieved by allowing review of the source code will depend on the application, but one thing that can be stated with certainty is that fragmented development is a short-term solution at best. 🖥️



Open Versus Closed Software Code

by Mark Evans

This article discusses the relative advantages and disadvantages of open versus closed software code. It also discusses an intermediate classification: “reviewable” code. No firm conclusions are drawn, but various considerations are outlined. This may help to determine the appropriate decision in a given situation.

Closed code may be appropriate when the problem being solved is well defined and static. This may apply to an algorithm to perform a complex but standard function, for example, an efficient sort algorithm. Here the expectations of what is to be performed are clear. One can specify what fields are to be sorted, which are primary criteria and secondary sort criteria, and whether the sort should be ascending, descending or ordered in a specified non-standard criteria.

Many general purpose software programs contain specific functions that perform well defined tasks. The input variables and output variables are well defined and the relationship between them is unambiguous. The Sumproduct function in Excel is a good example. Everyone knows what the function is supposed to do. It is a straightforward operation, and the user can reasonably expect the function to always work properly. There is not a reasonable need for open code in this case.

There are different degrees of open versus closed code. At the extreme, closed code could be defined as that where the client has no access to the source code for the system. At the other extreme, open code can be taken to mean the source code is delivered to the client with the expectation that the client will modify the code without restriction. An intermediate state is to make the source code available to the client for reference purposes, but not with the expectation that changes would be made. Perhaps this could be referred to as “reviewable” code.

Reviewable code allows the client to understand how the output of the software is determined. This can help avoid the situation where output is assumed to be sacred because the computer produced it.

As an application becomes larger, more general and dynamic, open code is more appropriate. Consider for the moment, actuarial modeling software. There are potentially infinite product variations, a wide assortment of approaches to setting demographic assumptions, various company methods for accounting for expenses, etc. It is impossible for a software vendor to anticipate all these possibilities. One solution is for the vendor to enhance the code as needed to provide new capabilities. This has problems, however. First, the vendor may have requests from multiple clients at the same time that may not be possible to meet. Second, even if the vendor can deliver the modifications to the software in a timely fashion, testing is problematic. With multiple possible outcomes as a function of an incredibly large number of inputs typical of actuarial functions, many of which drive decision-points within software, often based on subtle changes in variables being modeled, it seems unwise for a company to relegate this task to a “black box” process.

With open code the clients can see exactly how various processes are being performed. They can enhance the code if necessary, according to their internal priorities. Testing is improved for a couple of reasons. First, review of code can help understand different possible logic branches and lead to a more efficient test matrix. Second, if a test produces improper results, having access to the code makes it easier to understand a given result.

There are some definite disadvantages to an open approach. The client may have limited knowledge of the overall architecture of the



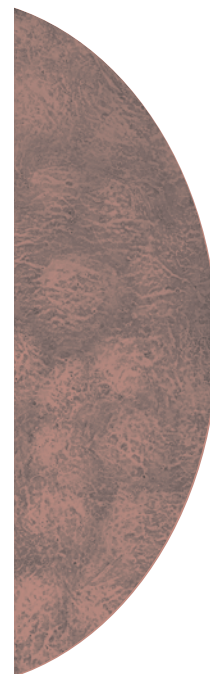
Mark Evans, FSA, MAAA, is vice president and actuary with AEGON USA Inc. in Louisville, KY. He can be reached at mevans@aegonusa.com.

system and not realize that a change made in one section of code has ramifications elsewhere in the code. Once the system is changed, it may no longer be practical to apply vendor supplied upgrades. If a client buys a system with the intent of altering it to that client's particular needs, may lose the benefits of vendor supplied upgrades. This means the client needs to establish and maintain the expertise to maintain the system going forward. This is not necessarily a bad thing as this approach has been executed successfully, but one should recognize the implications.

An intermediate approach is for the vendor to provide program "hooks" to facilitate migrating to future vendor upgrades. This means the "core" functionality of the vendor software is not altered by the user, but logic modules specific to the users, needs can be attached to the vendor supplied software. This requires careful design by the vendor, but if properly implemented, provides a balance between client flexibility and vendor maintenance and upgrade capabilities that may prove far more practical than either a totally open or totally closed approach. Making the "core" code reviewable has particular advantages here, because the user may be able to better understand how the hooks interact with the core. This approach has been successfully employed in practice. Vendor concerns about maintaining the viability of various client developed modules, however, may retard advances the vendor may otherwise be able to introduce. Advances in software robustness or run time efficiencies may not be applicable to the core software as a result. Also, as users make their own enhancements, the vendor is out of the loop, even in those cases where multiple clients (or even multiple users for one client) need the exact same functionality and each duplicate the same effort where the more desirable approach globally would be for the vendor to provide the solution. Also, vendor

upgrades will be used by a larger population, increasing the probability that any bugs will be identified quickly.

To a large extent, this is situational. A vendor with an unusually talented staff may be able to maintain a closed system efficiently and provide frequent upgrades to meet users' needs. Even in this case, however, the vendor, and in turn, the client, are exposed to key person(s) risk. On the other hand, a client should not embark on an effort to modify an "open" system without proper resources currently and the commitment to maintain those resources in the future. It too, may be exposed to key person(s) risk. A closed system may be more appropriate for a stable environment such as a valuation system for legacy products. An open system may be appropriate for a rapidly changing environment such as product development for recent types of variable annuity minimum guarantees. 🖥️



Open or Closed Code: A Response to Mark's Article

by *Nazir Valani*

Mark's article is very interesting. I would like to provide a perspective from a consultant who has used both open and closed code software for many years.

I will end up with the conclusion that is the opposite of Mark's (i.e., as an application becomes larger, more general and dynamic, closed code may be more appropriate).

We have had excellent experience using closed code software. Of course, there have to be a number of things in place for **closed** code to work well:

- The software must be designed with user flexibility in mind.
- The vendor must be capable of making any required changes in a timely manner.
- If the vendor allows the users to write its own logic, then user written code must be isolated from the closed vendor code, maintaining full compatibility on upgrade. This means treating user logic as data.
- Code modification process must be centralized and run in a fully professional development environment.
- Extensive testing must be conducted using multiple-user data from multiple client companies to minimize the likelihood of bugs in the software.

I have used a few open code systems as well. My experience with open code software has not been so good. In the long run, we seem to run into the following issues:

- The users have made many customized changes to the software. In addition, the vendor has also made numerous changes to the code at the same time. Upgrading to a new version of the software takes significant time, resources and money. It is true that the common code (e.g., locked library functions) is easily converted to the new version. However, the customized code can be a significant effort. Many times, there are so many changes in the code that it is not practical to upgrade the software. In these cases, the company ends up being stuck on an obsolete version of the software.
- Usually the code changes are left to a handful of individuals. Any staff turnover of the key individuals can be of significant concern. As the application gets more complex, the learning curve to be efficient is much longer. Also, the understanding that a change made in one place may impact other calculations takes time and expertise.
- Actuaries do not always make the best programmers. I have seen some bad code where an actuary is under pressure to produce the results (and may not have the expertise of a professional programmer). The code changes are made in a rush. The programming is "quick and dirty" to get the job done. In the longer term, this will be an issue. The efficiency of the code including the speed of execution may be impacted. Programming should be left to professional programmers (some may be actuaries as well) who are not responsible for producing the results.



Nazir Valani, FSA, FCIA, MAAA, is president of Valani Consulting Inc. in Toronto, Ontario. He can be reached at nvalani@valaniconsulting.com.

- Controls seem to be much more of an issue with open code. This can be an issue with Sarbanes Oxley compliance requirements. Is there a new version distributed every time a user makes a change? Who is responsible for peer reviewing the change made? Who does the testing to make sure that you are getting the expected results and that the change is not having any unintended consequences? How are the changes coordinated between different users making the changes? Ideally, you want different individuals performing the different tasks.

My experience has indicated that as an application becomes larger, more general

and dynamic, **closed** code may be more appropriate.

Admittedly, whichever way you go is dangerous. If you go the closed code route, you are reliant on the vendor. You better make sure this is a vendor you can trust. Ask around, look at the track record. If you go the open code route, you will have problems that only increase through time. Initially, setting up and maintaining a professional programming and testing environment, and ultimately incurring significant expenses of a full conversion process to avoid being locked into an obsolete architecture. 🖥️



Noteworthy!

Phil Gold is the pioneer featured, as part of the PIONEERS: Real Life Examples series, in the August/September 2006 issue of *The Actuary*. In the article title, "Something You can Use", "...Phil Gold is fascinated with technology and he takes every opportunity to make it work better for others!" Gold talks about his desire to improve the quality of other people's jobs, his innovations and the importance of working with a good team.

For a hard copy of this article, pick up the *The Actuary* magazine. You can also find this article [online](#).

Annuity Calculations Using SQL

by Sheila Silva

The power of SQL to perform bulk calculations has been firmly established within the IT community for many years now. However, actuaries are only now just beginning to move beyond simple spreadsheets and traditional programming languages to accomplish our “everyday” tasks. In this article I will focus on using SQL to value annuities. A simple annuity is used below solely for expositional reasons; the user can build upon this methodology to add decrements, offsets, varying interest rate curves, etc.

What Is SQL?

SQL, or Structured Query Language, has been used to manipulate data within relational database management systems, such as Oracle®, DB2® and SQL Server®, for decades. While there exists an ANSI standard for SQL that is supported by most vendors, each vendor has added its own bells and whistles on top of the standard to create versions for use within their own products. I will use the ANSI-92 SQL, a widely-supported standard within the industry, so the user can test out the queries without modifications.

SQL code is quite readable. The following query retrieves the claim ID and reserve fields from a table called Claims:

```
SELECT ClaimID, ReserveAmt FROM Claims
```

SQL also allows the joining of tables. This in turn allows for data to be both stored and retrieved efficiently. This is an improvement over the common practice within our profession to create enormous spreadsheets that contain all data within one, big table. The following query retrieves payment information for each claim from the tables Claims and Payments:

```
SELECT Claim.ClaimID, MAX(PaymentDate)
AS PaidThru, SUM(PaymentAmt) AS TotPmts
FROM Claims JOIN Payments ON
Claim.ClaimID=Payments.ClaimID GROUP BY
Claim.ClaimID
```

In the above query, I’ve used another feature of SQL—aggregate functions. Here, I’ve specifically used the MAX and SUM functions. They operate as they do in Excel, but because a database is designed for aggregation, they perform more efficiently than they do in Excel. (Without going into a comprehensive review of SQL syntax, it suffices to say that queries that use aggregate functions require a GROUP BY clause.)

SQL allows for the creation and use of temporary tables, which are “virtual” tables derived from “real” tables. They are identified by the use of the pound sign (#) at the beginning of the table name, and are created with a SELECT...INTO statement:

```
SELECT ClaimID, YEAR(DisDate) INTO
#TempTable FROM Claims
```

As actuaries, we are all too familiar with the multiple copies of spreadsheets that we create, each slightly different to accommodate a particular use (i.e., rating, reserving, scenario testing, etc.), but all containing the same source data. Months later, we often find it difficult to remember the differences between MonthlyData1.xls, MonthlyData2.xls and MonthlyData3.xls, let alone reconcile them. Temporary tables allow for these “copies” of data, which are available for use in queries during the life of a database session, and are automatically destroyed at the end of that session. (As an aside, we note that SQL also allows for views, a more permanent construction of a table-derived-from-other-tables.)



Sheila Silva, FSA, MAAA, is the second vice president of Actuarial Information Technology with Smith Group, a disability reinsurance risk manager and consultant. She can be reached at SSilva@SmithGroupRE.com.

Review of Annuity Calculation

Most actuaries are quite familiar with the standard annuity formula. Here I present a simplified version of one used for disability insurance annuities:

$$GB*(1+i)^{-1}*(1-q_1) + GB*(1+i)^{-2}*(1-q_1)*(1-q_2) + \dots + GB*(1+i)^{-t}*(1-q_1)*(1-q_2)*\dots*(1-q_t)$$

Where:

GB=monthly gross benefit

i=interest rate (for this simplified example, we will assume it is based on disability year)

qs=claim termination rate for period s

Each of the terms in the above formula can be broken down into three components: the benefit amount, the interest discount and the survival rate. The first two components lend themselves easily to SQL statements; for example, the following query would compute the value of an annuity of \$1/year from a table of annual interest rates (assuming the table begins with the year 1990):

```
SELECT (IntYear-1989) AS DurationNum,
SUM(POWER(1+IntRate,(1989-IntYear)))
FROM AnnIntRates GROUP BY (IntYear-1989)
```

However, the survival rate components, since they are a geometric progression, require some refactoring before SQL may be applied. A little exponential math helps solve the problem, with the following equation:

$$x*y=(e^{\ln x})*(e^{\ln y})=e^{\ln x+\ln y}$$

This equation allows us to convert a product calculation into a sum calculation, one of SQL's strengths:

```
SELECT DurationNum, EXP(SUM(LOG(1-
TermRate))) FROM TerminationRates GROUP
BY DurationNum
```

SQL's other strengths, temporary tables and the ability to join tables, allows us to create all the terms in the progression to achieve our result (this query assumes that I have created temporary table #AnnuityTable that simply contains claim durations 1 through N for each claim):

```
SELECT Claims.ClaimID,
#AnnuityTable.DurationNum,
GrossBen*POWER(1+IntRate,-
1*#AnnuityTable.DurationNum)*EXP(SUM
(LOG(1-TermRate))) FROM #AnnuityTable
JOIN TerminationRates ON #AnnuityTable.
DurationNum=TerminationRates.Duration
Num JOIN Claims ON
Claims.ClaimID=#AnnuityTable.ClaimID
JOIN AnnIntRates ON
YEAR(Claims.DisDate)=AnnIntRates.IntYear
JOIN #AnnuityTable A2 ON
#AnnuityTable.ClaimID=A2.ClaimID AND
#AnnuityTable.DurationNum>=
A2.DurationNum
```

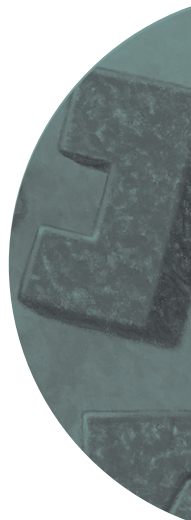
Note in particular here the join clause in bold-face. This join is called a self-join, since it connects the table #AnnuityTable to itself. However, this is not a one-to-one join. By joining each duration in #AnnuityTable to all the durations less than or equal to it, we are able to create the series of annuity terms. Subsequently applying the SUM function to this progression gives us the result we were looking for.

Performance

Of course, clever methodology doesn't count for much if performance is lacking. Here is where the real power of SQL becomes apparent. Set operations are inherently faster than programmatic loops. To demonstrate this, I created a sample SQL Server 2000 database, with claim, interest rate and termination rate tables. That database includes a stored procedure (i.e., a set of compiled SQL statements)

(continued on page 20)





| The results are telling | | |
|-------------------------|-------------------|------------------|
| Claim Count | Time Elapsed–Code | Time Elapsed–SQL |
| 1,000 | 25 seconds | 2 seconds |
| 100,000 | 35 minutes | 3 minutes |
| 1,000,000 | 6+ hours | 29 minutes |

that performs the actions as described in the previous section. (Note: the queries described above were refactored a bit to optimize performance.) I also created a VB.NET program that performs the same annuity calculations, but using traditional programmatic loops (i.e., do-while, etc.). My sample database includes 1 million claims, so that the time comparisons would be meaningful.

As you can see, the stored procedure performed better than the traditional code in all cases, and was especially powerful when there were more claims involved. This performance difference can mean shortened monthly reporting cycles, and the possibility of more detailed analysis, because seriatim reserving would become less time consuming.

Conclusion

Claim data is increasingly stored in databases, where SQL is the language of choice. As actuaries, we have become accustomed to obtaining flat-file extracts of these databases as inputs to our reserving and pricing programs, as well as our ad-hoc spreadsheets. This practice is equivalent to reinventing the wheel—why not use the database itself as the actual input to our calculations? Better still, we can use SQL and the power of the database engine to perform these calculations. While I have presented a simplified model here, it can be extended to a variety of applications involving a host of other products.

A side benefit of making use of the existing databases is increased involvement by our industry in improving the quality of the data gathered. We are all aware of the saying “garbage in, garbage out.” One of the very reasons we work off of extracts is that we feel we have more “control” over input quality, by massaging the extracts before we use them. But we are doing our industry a disservice by side-stepping the real issue this way. By applying our thorough understanding of the business value of the information gathered in those databases, we further our own goals of accurate business knowledge and also improve the credibility of the entire industry. 🖨