SOCIETY OF ACTUARIES
MODELING SECTION

The Importance of Centralization of Actuarial Modeling Functions, Part 3

Implementing Model as Data Using Moody's Axis

By Bryon Robidoux

In part 1 of this article, I used software engineering principles to show that decentralizing models come with very high costs. I showed that centralization of a modeling department is a step in the right direction, but it isn't enough. The key to running a smaller, better, faster and cheaper modeling department is to focus on modularity and work product[1] reuse according to software engineering principles. Part 2 introduced the reader to the major components of DevOps and how it is the basis for actuarial modernization. But up to this point, there haven't been any practical examples of how to implement DevOps or what the end result would look like.

This article will focus on how to transform a Moody's Axis model that suffers greatly from the monolithic-system problem[2] to a model that has maximized data reuse and promotes the use of DevOps. This will be defined as a Model as Data (MAD), a model that has completely data-driven processes that will speed up the throughput of enhancements, improve testing, simplify production processes and make ad hoc runs easier.

AN INTRODUCTION TO MOODY'S AXIS

Non-Axis users may need a frame of reference for its two major components: E-Link and the dataset. E-Link's main goal is to manage the collection of the organization's models and orchestrate their execution. It has a very Windows Explorer feel. E-Link can be automated with scripts to externally manipulate datasets and customize orchestration using Axis Jobs and E-Link scripts, respectively. One of the most important enhancements to E-Link in the last year or so is Formula Link. This extension builds reusable libraries that can be shared among multiple models and E-Link scripts.

The dataset is the model itself. This is where 98 percent of the work is done. From E-Link's point of view, the dataset is like a big zip file full of Axis proprietary and user-created files. From within the dataset, the dataset's interface has its own subcomponents, such as batches, remote tables, datalinks and datalink tables. Batches instruct Axis to do calculations and other operations needed to manipulate the dataset. The remote tables point to data that live outside the model. Datalink batches transform the remote tables into datalink tables. Datalink tables are the internal tables the model uses for seriatim policyholder information and other data. The cell tables and projection tables are used to describe how a particular insurance plan or rider will be calculated.

IT'S ALL ABOUT THAT DATA

The dataset can be split into two fundamental pieces: data and code. This article will be focused on the data portion, while the

code will be addressed in the next article. One of Axis' powerful features is its import/export batch. On export it creates a turnaround document, which is a file that can be exported in one step and then reloaded into the program at a future step to populate with new data. This allows a user to dynamically create Axis objects without manual intervention and consequently means that a large portion of the dataset can be created on the fly. Examples of objects that can be exported are datalink tables, cell tables and projection tables, which means they can be treated like data. This will be defined as Calculation as Data (CAD).

Data Transformations

In the worst-case scenario, there are up to three places where data can potentially be transformed. The first is within the databases inside the insurance organization. The second is the SQL server instances that sit in the Moody's cloud. The third is within the dataset using datalink. If your organization is using AWS or Azure, they both remove the Moody's cloud layer. As data are transformed and moved, each layer in the data architecture increases the controls because the actuary has to validate that the transformations were successful. (When a production run breaks down over the weekend, there is nothing better than having to dig through three different locations and multiple people to figure out what went wrong. Those are the weekends I look forward to the most!)

All the data should be located in one place along with its transformations. This location should be outside the dataset to create a one-stop-shop for diagnosing data issues, to simplify debugging and to reduce the monolithic-system problem by maximizing data reuse. The data should be transformed until they can be loaded one-to-one with a resulting datalink table, cells table or projection table that they will inevitably populate. Never read directly from a database table. Always read from a view or materialized view to promote good data encapsulation protocols.

Using Files

The actuary should not make the dataset a dumping ground. Even though Axis allows it, no files of any kind should be stored within the dataset. (This is a huge source of potential error if people use outdated files, and it can cause confusion during reporting cycles.) Loading files from the dataset requires manual intervention and makes the data-driven processes clunky at best or moot at worst. If external Excel model results are used to populate the dataset, they should first be stored in the corporate database along with all other data so the results can be reused easily by multiple parties and be included in the automated data assembly process. This removes the potential of putting production data in email and using the incorrect version for a production process. How many times has this caused problems in your organization?

CONTINUOUS DATA INTEGRATION

Continuous data integration allows the same hydration routines to be reused to dynamically create models for production calculations, attribution analysis and ad hoc analysis. The fully data-driven, flexible model will encapsulate and abstract away the details so these calculations can be uniformly handled, thereby eliminating the manual setup usually required.

To implement continuous data integration, automation is required. Automation might include a simple interface—external to the dataset—to select data for any activity, such as a testing or production run. Behind the interface would be all the mechanics and metadata required to assemble the correct inputs, hydrate the model and execute a run. All interfaces should be configurable and script driven, so they can be parameterized and executed repeatedly without human intervention.

The valuation team should never have to touch the dataset during a production cycle. Better yet, there should be no manual processes required during any reporting cycle to produce results from execution to final report. If such processes are required, it implies that at least one manual touchpoint is likely to produce errors, so controls will also have to be created and productivity will be murdered. All required materials should be staged and ready to go **before** that day occurs. Any processes that pull the latest market data or the like should be viewed as part of the model and automated accordingly. Errors cause the model to be rerun, deadlines to be missed, weekends to be lost and the model execution's cost to rise.

CONTINUOUS DEPLOYMENT

To continuously deploy changes to a model, the data need to be broken down between different environments such as development, quality assurance and production. It is critically important that database architecture be nearly identical for all three types of environment so that only the remote tables have to be updated to make all processes work. The process of deployment consists of nothing more than appending the turnaround documents created for the new features to the staging environment's turnaround documents. If any GUI is required for deployment, the productivity and robustness will suffer immensely.

The ultimate goal of a MAD is to create an environment in which all runs and the hydration of their data can be orchestrated through E-Link scripts. These scripts should depend heavily on Formula Link libraries for building reusable components with maximum logic reuse. For ultimate automation, these libraries should be data-driven so every aspect of running the model can be dynamic and configurable.

CONTINUOUS TESTING

The testing team will want data categorized for different types of tests, such as positive[3] and negative.[4] There is a tendency to use only positive testing in a modeling department, especially when the dataset has a million manual processes to import data

A get-it-done attitude no longer works. … The focus needs to change to a get-it-done-better attitude.

for use. No one will want to go through the time, effort or trouble to load different sets of data. Or worse yet, teams are afraid to use testing data because such data may accidently slip into the production model! It is important to realize that DevOps promotes using production-like environments for testing and development, but that doesn't imply that only production data should be used in all environments. This leaves many potential errors uncovered. (There is nothing worse than a vague or confusing error message in production that sends the valuation team on a wild goose chase with delivery timelines quickly approaching—especially when the issue is a simple input error that an intelligent error message could have easily pointed out.)

Continuous testing is only possible with continuous integration and continuous deployment. With these capabilities, the testing team can hydrate the model with any data they need to verify that the model is fit for production. Tiny models that are specifically targeted at the enhancement or fix being implemented can be created on the fly. This immensely improves the testing that can be performed by reducing errors, allowing the testing teams to receive feedback in minutes instead of hours or days and allowing features to be implemented within the time frame of a single Agile sprint. There is no possibility of the wrong data being used because the data are controlled by the well-tested environment. This means all kinds of tests can be run to verify that the model will handle bad data gracefully and good data properly.

TELEMETRY

As the data are assembled and transformed so they can be placed directly in the model, logs, error checking and efficiency metrics need to be captured and retained. They will provide immediate feedback on the health of the system, and they can help actuaries and IT find places in the processes where weaknesses and bottlenecks can be improved. All this enhances the turnaround time for results, so decisions can be made faster with better information.

BACK TO LIFE! BACK TO REALITY!

Now that the ultimate modeling platform has been described, where does reality set in and thwart the vision? It all falls apart with the batches! Why? you might ask. As of July 2019, the last time I used Axis, the batches cannot be exported from the dataset so that all the fields of the batches can be externally manipulated. In other words, Axis has no turnaround documents for the batches. The only fields that can be manipulated externally through Axis Jobs and E-Link scripts are in the dataset parameters, but fields such as report location and override set are not located in the dataset parameters. This means that from the user's perspective batches cannot be updated without using the dataset interface. This leads right down the manual processes' and manual controls' rabbit hole that should be avoided. **It is important to be able to export all batches so the model's behavior can also be data.** Thus, the dataset's behavior can be completely and dynamically manipulated. This is defined as Behavior as Data (BAD).

INFRASTRUCTURE AS CODE

Having Axis objects and batches as data—that is, CAD + BAD—is required to effectively create a MAD-compliant model. MAD is synonymous with the infrastructure-as-code (IAC) concept within DevOps. IAC means that any part of an application's environment and its versioned controlled components can be provisioned and set up in an automated fashion with no manual intervention. To generalize this to actuaries, the spreadsheets, Alteryx scripts and so on should also be contained in source control. Source control, such as GitHub, does not allow comparison or track changes of binary files such as spreadsheets. This is another reason to discourage the use of spreadsheets, unless the organization wants the added complexity of coordinating Incisive with GitHub.

CONCLUSION

This article has focused on reducing the monolithic-system problem by making the model and its processes data-driven so they are DevOps complaint. Model as Data makes the production, audit and controls teams more confident in the model they receive because it is designed specifically for continuous integration, continuous deployment and infrastructure as code. Therefore, it allows all aspects of production to be tested *before* the production cycle.

Developing a data-driven model that is compliant with MAD vision is no simple task. It requires a massive amount of infrastructure, which in turn requires a massive amount of coordination between IT and actuaries. To make this successful, it is important to remove the silos between these professional groups and "cross-pollinate" training and ideas. This will require cultural changes in how actuaries and IT work together.

At some point, the organization needs to decide where the complexity is going to reside. There is no such thing as a free lunch, and the complexity has to reside somewhere. Will it be in the automation of the processes or in the operation and coordination of people and their manual touch points? The latter may seem a comfortable choice because the complexities are all surface level and observable, but it will kill the opportunities to create economies of scale. Every new task will require a new person, eventually leading to more managers orchestrating the processes. The organization will get so large

and hard to move that it will be crushed by its own weight. As regulators and auditors expect more and faster results, organizations will beg regulators to kick the can down the road because they are overwhelmed with change. A get-it-done attitude no longer works because all the late nights and weekends are already being consumed with the current processes. As people are pushed harder, the technical debt will grow faster and more abundant, which is actually counterproductive.

The focus needs to change to a get-it-done-**better** attitude. This means constantly analyzing the processes for bottlenecks and brittleness, learning better approaches from other disciplines, finding the similarities and differences between processes and exploiting these similarities to create economies of scale. This can only be done if the automation of processes choice is made. Actuaries need to realize that the only way forward is to embrace technology and use it to its full extent. Just producing numbers may have been acceptable in the past, but now it is just as important to understand how to build the most efficient and maintainable processes possible. If you are not willing to do it for your organization, then do it for the health and viability of your profession!

If organizations and actuaries are willing to brave this new world, the modeling and valuation departments will become much faster. Their development methods and processes will become compliant with IT approaches to development. Development, testing, production and special analysis will become much easier. Late nights and herculean efforts should subside, which will increase morale, productivity, the confidence in the model's results and the organization's efficiency. ■

Bryon Robidoux, FSA, CERA, is a lead and corporate actuary, Actuarial Transformation, at The Standard. He can be reached at *bryon.robidoux@standard.com.*

ENDNOTES

1 A work product is any logic, data or data transformation that has the potential of being used in multiple places within the organization.

2 The monolithic-system issue was defined in part 1 of this article. It comprises the problems that exist from locking away all the logic, data and data transformation in the model so they can't be reused elsewhere. This is counter to sound software engineering issues and leads to duplication of effort on many fronts.

3 Positive tests use data that are within valid ranges to verify the system works as expected.

4 Negative tests use data that are outside valid ranges to verify the systems will fail where expected and have meaningful error messages.