Article from

The Modeling Platform

April 2020

The Importance of Centralization of Actuarial Modeling Functions, Part 2

DevOps—The Path to Actuarial Modernization and Consolidation

By Bryon Robidoux

The first article in this series stressed that consolidating the actuarial modeling department was an important and worthwhile initiative. But consolidation doesn't solve much if the redundancies and complexities of the modeling department are not reduced in the process.

It was suggested that software engineering practices have many answers to our modeling problems, especially the monolithic system[1] issue, but there was no mention of what concepts were important or how to get started. This article will fill that gap by introducing IT DevOps and other software engineering principles and their application to the current actuarial modernization and modeling department consolidation.

CHANGE, CHANGE, CHANGE AND MORE CHANGE

Let's pause for a second and look at the actuarial modeling and processes landscape. The amount of change required of the insurance organization depends on the type of business written and where it is sold. In 2017, there was VM-20; in 2019, there was VM-21; and in 2021, there will be LDTI (long duration targeted improvements), and IFRS 17 is coming. Not to mention, interest rates have been excruciatingly low and the S&P has been on the rise for the last 11 or 12 years, so the products are getting more equity features to stay competitive.

Auditors and regulators are mandating that senior managers be able to attribute and explain changes to demonstrate confidence that their organization can properly manage its risk. Given the speed with which the playing field is changing, senior managers need to quickly and confidently do what-if analyses to make more informed decisions to stay ahead of their competitors. They need information quickly, which requires processes to be accurate, streamlined and efficient. They can't wait weeks or months for actuaries to update their spreadsheet processes for a decision that needs to be made in a week, or a day or less. As changes accelerate, change management and handling complexity become paramount.

DECENTRALIZATION—THE NASTY TRUTH

The previous article mentioned that decentralizing models is a bad practice and should be avoided, but it failed to recognize the driver behind the behavior. The real motivation for decentralization is to reduce the complexity of models so they are more maintainable and easier to understand.

The desire to keep things simple is a worthy cause, but decentralization is trading model complexity for operational complexity. There is a great book on this topic originally written in 1975

called *The Mythical Man-Month*.[2] The premise of the book is that it is a mistake to think that if one developer can do the job in one year, then hiring 12 developers will get the job done within a month. It explains that this is not possible because people can't learn complex systems instantaneously. Even if this were the case, communication among people and teams causes the development to slow to a crawl because everyone needs to coordinate. The book recommends coming up with a team of specialists who work together to accomplish an overall goal. These specialists should complement each other in such a way that they can maximize the ability to work independently, reducing operational complexity.

KEEPING MODEL COMPLEXITY UNDER CONTROL

In software engineering, refactoring and unit testing are performed together to mitigate model complexity. Refactoring is the practice of cleaning up the models to make them easier to maintain without changing their behavior. Unit testing is writing small, fast and single-purpose tests to verify the software is working as expected. Great books on these topics are *Refactoring: Improving the Design of Existing Code*[3] and *Working Effectively With Legacy Code*.[4]

The less frequently refactoring is performed, the faster the model's complexity will get out of control. If anyone mentions that there needs to be a project to refactor the code base, then this is a good sign that the development practices and standards of the organization should be revisited. Refactoring should be akin to cleaning up the woodshop at the end of each day's shift so that everything is clean and organized for the next day.

HANDLING CHANGE OUTSIDE THE ACTUARIAL PROFESSION

How do Facebook, Amazon, Netflix and Google (FANG)—along with other large technology organizations—handle hundreds of developers confidently, making many changes to their code on a daily basis and not suffering from the same model and operational complexities that actuaries suffer from? The answer is DevOps. It is a framework and guidelines on how to efficiently handle rapid change with confidence and reliability. It allows developer teams and operation teams to work closely together building robust processes and software systems.

DEVOPS

One main goal of DevOps is to shorten the deployment time of fixes and enhancements for complex software systems. It borrows a lot of its methodologies from lean manufacturing.[5] Even though it was conceived from manufacturing circles, there is no reason that actuaries should not exploit it for their needs! *The DevOps Handbook*[6] is a great book to get up to speed on the topic. To promote speed and reproducibility, automation is at the heart of DevOps, but it is bigger than that.

There are several components to DevOps, such as microservices, continuous testing, continuous integration, continuous delivery, continuous deployment, infrastructure as code, telemetry and continuous feedback, which will be discussed next.

Microservices

With DevOps, the collaboration can happen at such a fast pace because each team works to build microservices. (I will take a little liberty in describing microservices.[7]) For actuaries, a microservice can be thought of as just a single-purpose library. Microservices allow developers to work independently without trampling on each other. They contain application user interfaces (API), which are interfaces that encapsulate the details of the implementation behind a barrier.

The interfaces have contracts, which are called preconditions and postconditions, that describe the output of the services based upon the domain of the inputs. As long as everyone writes codes based upon these contracts, there is no reason to worry about the details of implementation. This greatly speeds up development, because it reduces dependencies among components in the model.

Microservices should be loosely coupled but have a tight cohesion, which means they should be able to communicate with each other, work independently and be singularly focused. The problem with monolithic systems is they have tons of dependencies that lead to tight coupling and loose cohesion of all their components. This leads directly to a system's complexity and the desire to decentralize it.

Continuous Testing

For each unit of work within a microservice, a unit test is made to verify that it operates as expected. These automated tests are small, fast, singularly focused and should run in milliseconds. They should not consume external resources or write to external locations, such as files or databases, so that they run very efficiently. They should be able to run locally on the modeler's local machine or on a server. This allows the developer to continuously run thousands of tests to get immediate feedback and quickly diagnose problems. Running a few sample policies is too slow and too little coverage. Running all policies on the grid doesn't give immediate feedback or good diagnostics on potential issues.

Once the enhancement passes all the unit tests, the changes should go through automated user acceptance testing (UAT). These should also be fast and plentiful, but they are usually larger, less granular tests. They would be designed to test the microservice API and its larger logical units. As stated in test-driven development (TDD), all the unit tests and UATs should be created before a line of code is ever written or modified so that the design of the tests is part of the design of the model. It is only after hundreds or thousands of the very fast automated tests

Monolithic systems naturally lead to Waterfall project management no matter how good the intentions are to go Agile.

have been run that more manual exploratory tests should even be considered.

Continuous Integration, Delivery and Deployment

As many developers are making changes throughout the day, the changes need to be continuously integrated into the master branch. If there are too many changes or the changes are too big, merging them can be time-consuming and difficult and can potentially produce instability. Therefore, each modeling task should be small and singularly focused to provide continuous delivery of new features multiple times per day.

Once the code is delivered, it can go through its last round of reviews and approvals. According to *The DevOps Handbook*, the reviews and approvals should not be delegated away to outside committees. The greater the distance between the committee and where the work is actually performed, the less familiarity there is with the changes and the slower the approval process will be. It is actually recommended to follow extreme-programming practices, which advocate for dual modelers working together on each task. This method has been shown to be quicker and more thorough than a committee approach, because the modelers help each other arrive at a better solution and spot potential issues faster. Once the approvals are passed, then the code can be automatically deployed into production.

INFRASTRUCTURE AS CODE

Infrastructure as code is the concept that all aspects of the model and its configuration are in source control, such as GitHub. This gives the ability for anyone to download the model and all its dependencies and quickly get any deployed version running and reproduce results. If things do go awry, the previous version can be brought back quickly with no manual intervention or setup time. For actuaries, this would include all the work products, such as spreadsheets and other items required to feed the model. This allows any part of the production environment or processes to be reproduced from beginning to end.

Spreadsheets are just ad hoc little programs that are mainly doing calculations and data transformations. They are manual process touch points that are cumbersome, error-prone and a major source of technical debt.[8] It would be much more robust to replace these with more traditional software applications so that the production processes can better follow DevOps principles.

TELEMETRY

Telemetry is monitoring and logging the model by recording data on all mission-critical aspects of its behavior. This allows problems to be addressed quickly with little or no downtime. Items to monitor are run times of all the intermediate processes and distributions of different input variables, crucial intermediate variables and output variables. By keeping the statistics, everyone can receive continuous feedback and learn ways to improve the processes and models. Machine learning and reinforcement learning can be used to monitor logs and detect errors faster, which will speed up response time of dealing with issues.

CONTINUOUS FEEDBACK AND LEARNING

In order for organizations to improve their models and operations, they need to be constantly learning from both their past successes and their past failures. This is not possible without continuously monitoring the health of the models and the supporting processes.

The problem with a monolithic model is that all the pieces have to come together in order to get a functionality to work. It might take days, weeks or months to get all the pieces assembled depending on the size of the enhancement. The feedback on all the issues does not come until late in the development cycle. At this point, the enhancement is promised to senior management, and herculean efforts are required to get it all done. The enhancements are often brittle to boot.

This is why it is important to create tasks that are small and singularly focused—so that the feedback on potential issues comes as early as possible in the development cycle. The later the problems are realized, the more expensive they are to fix. This is why there has been a strong movement of Agile project management over Waterfall so that everyone can get immediate feedback and fix problems sooner. Monolithic systems naturally lead to Waterfall project management no matter how good the intentions are to go Agile.

FUTURE ARTICLES

Now that DevOps has been introduced, the following two articles in the series will get away from theory and get to the practice of implementing DevOps using Moody's Axis. Part 3 will address continuous integration, continuous delivery and infrastructure as code by creating a data-driven dataset that can be generated on the fly. Last, Part 4 will implement DevOps in code using Axis's formula link, formula tables and third-party DevOps tools to showcase all the principles in this article. With these detailed case studies, it will give actuaries the ability to start implementing DevOps in their organizations.

CONCLUSION

Given all the regulatory and accounting changes, such as LDTI and IFRS 17, the actuary has been asked to make a lot of changes in recent years. Even though the intentions of the regulators and accounting standards are to help produce stronger insurance companies and to better track changes, the current practices of actuarial modeling and processes have had a hard time coping with the tidal wave of change. The changes are too fast and the complexity too large for actuaries to brute force their way through them anymore. DevOps is the paradigm shift needed to better cope with change and change management.

The main aspects of DevOps are microservices, continuous testing, continuous integration, continuous delivery, continuous deployment, infrastructure as code, telemetry and continuous feedback. Each one of these concepts plays a crucial role in improving the actuaries' change management capabilities. By following the DevOps best practices, actuaries will be able to create smaller, better, faster and cheaper modeling and valuation departments. The herculean efforts required to get through production cycles and do what-if analysis will be greatly reduced.

Actuarial modernization should be more than moving to a new modeling vendor or software package. The support from our vendors is critical, but modernization is bigger than them. It is about changing how actuaries work and their culture by embracing DevOps and making the practices commonplace. Actuaries are not really modernizing if they are not incorporating DevOps practices in all their work. Replacing spreadsheets should be the first focus of all modernization efforts, because there is so much to gain. Spreadsheets in processes are like cockroaches. There is never just one, and it is expensive and difficult to get rid of the infestation!

With all the potential changes and unknowns on the horizon, it is important that actuaries incorporate DevOps practices sooner rather than later. ■

Bryon Robidoux, FSA, CERA, is an actuary ALM at Reinsurance Group of America. He can be reached at *bryon.robidoux@rgare.com.*

ENDNOTES

1 The monolithic system problem is when all the work products needed for the model are modified and/or stored in the model such that the model will not run or produce accurate results if all the components do not exist.

2 Brooks, F.P. 1995. *The Mythical Man-Month*. Boston, MA: Addison-Wesley.

3 Fowler, M. 1999. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley.

4 Feathers, M.C. 2005. *Working Effectively With Legacy Code*. Prentice Hall.

5 From Wikipedia, lean manufacturing or lean production is a systematic method originating in the Japanese manufacturing industry for the minimization of waste within a manufacturing system without sacrificing productivity, which can cause problems. *https://en.wikipedia.org/wiki/Lean_manufacturing*

6 Kim, G., J. Willis, J. Humble, and P. Debois. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability and Security in Technology Organizations*. 2nd ed. Vol. 2. T Revolution.

7 Library vs microservice. static void, March 7, 2017, *https://blog.staticvoid.co.nz/2017/library_vs_microservice/* (accessed March 12, 2020).

8 From Wikipedia, technical debt is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer. *https://en.wikipedia.org/wiki/Technical_debt*