Article from

CompAct

April 2017
Issue 55

Coding Standards and the Efficient Model

By Brody Lipperman

For most actuarial programmers, a lot of thought and effort is put into making their models run faster. Gains in speed are easy to measure, look good on reports, and either save money by requiring cheaper hardware to run, or gain value by allowing the actuaries to run more models and get more data. Run time, however, is not the only way to make your models more efficient.

In the normal life cycle of an actuarial model, there are thousands of hours put into developing, enhancing, testing, explaining, documenting and validating. The total cost of the human capital used for these models vastly outstrips the cost of the hardware required to complete a model run in an acceptable amount of time. Usually, very little attention is paid to any gains in these areas because they are very hard to quantify. If I spend 10 hours cleaning up code and documenting my model, does that save 10 hours of time down the road as various other people have an easier time of understanding the model? While the benefits of this type of efficient model are often hard to quantify, they are without a doubt just as valuable, if not more so, than the benefits of run time improvements.

> One of the best ways to improve a model's overall efficiency is to develop a set of detailed coding standards. ...

One of the best ways to improve a model's overall efficiency is to develop a set of detailed coding standards for your modeling team. These standards should address stylistic considerations, function use, documentation rules, and any other aspects of model coding. If done properly, this should allow your model to be easier to read and understand, decrease the amount of time required to make changes, reduce key-man risk, and reduce coding errors. It will require a change in the mind-set of the team, and buy in for a shift in personal responsibilities.

A good set of coding standards should address most issues that will arise when writing code. While bracket placement and indentation length might not seem overly important at first, uniformity in the presentation of code allows users to focus on the content of the code, rather than being distracted by stylistic differences. The standards should also address naming conventions. Today's languages have typically removed size constraints, so developers should strive to avoid abbreviations when possible, but there should be rules on what abbreviations to use if necessary (so that users will understand that ANN is short for annual, not annuity). In setting up the standards, it is important to set rules for most coding situations, even if they are arbitrary (for example, a function should be no longer than 30 lines).

While exceptions to the standards can be approved by model owners, they should be well thought out and documented. If a code change will decrease run time by 10 percent, but increase the complexity of that section of code, attention should be paid to both the benefit and the downside. If the model is run overnight, would the 10 percent speed increase have any noticeable benefit? Or if weighed against the fact that the more complex code would take more time to explain to end users, could only be modified by a specific set of coders, and increases the risk of future errors, is it worth it? This type of decision-making should be brought up for each exception to the coding standards.

Each model (or subset of the model, for more complex models) should have a model owner. While the model owner will have the final say over content, they should not be the only person that validates that the coding standards are maintained through the model. Each user should have shared ownership and responsibility for the entire model. Frequent code review meetings are a fantastic way to foster this responsibility. Members of the team present code that they have been working on, and get feedback from other members of the team. This allows all members of the team to scrutinize the code and ensure that it conforms with the coding standards. It also allows less experienced developers to learn from their more tenured counterparts. Knowledge is shared across the team, both in coding methods and content of the model. This will reduce key-man risk, as each developer should understand any new pieces of code in the model, as they will be reviewed in these meetings.

It is important to encourage all developers to provide feedback, as there can be a tendency to have "experts" in various areas of the model. These experts will receive less scrutiny with their code reviews, and as a result, will typically have more mistakes in their final product. Other developers will also tend to defer to the expert when questions about their areas of code arise. This can reintroduce key-man risk, and potentially bottleneck future change requests.

Coding standards should also cover reusability aspects of the code. Any formula that is repeated in the model, should be converted to a function. By centralizing the code, developers can reduce the amount of time it takes to make any future changes. They can also reduce future errors that would be caused by changing a calculation in one section of the model without modifying the same calculation in another section. Having well-defined function names can also make reading the code easier for end users. If a user wanted to understand what all is included in an AV calculation, they can look at the code and see the calculation includes COI charges. They don't necessarily care that the COI function calculates a NAR after premium and loan interest is taken out. Functions allow users to absorb as much detail about the code as they want, while still being able to dive into each function if they need more information.

The final aspect of the coding standards should be rules around types and goals of documentation. Since all modern languages resemble English, the model code should be self-documenting. Variable names should be descriptive enough to be easily understood without any reference, and most users should be able to follow the basic logic constructs (if then, for loops, etc.). Code should be written in such a way as to reduce the complexity of each section of code as well. If the developer needs to document end points for If Then statements or for Loops, then they should attempt to break the code into smaller, more digestible blocks of code. The goal of documentation, then, should be to explain why the code does what it is doing, instead of what the code is doing. This will allow future developers and end users to understand the choices the developers have made, and allow them to follow the whole model easier. Any documentation included in the code should be short and concise, anything longer than two sentences should be moved into a more formal document.

A strong set of coding standards and the proper team mindset can greatly reduce the amount of manpower required to maintain a model. By setting up rules and guidelines, developers are forced to consider methods that are easier to understand for future developers or end users and can create a better overall product. These standards can also help spread knowledge, responsibility, and ownership throughout the team, leading to a stronger, more flexible organization. The end result should be a model that is easy to maintain, easy to understand, easy to validate, and easy to manage. ■

Brody Lipperman, FSA, CERA, MAAA. He is a Lead Actuarial Developer with FIS. He can be reached at *brody.lipperman@fisglobal.com*