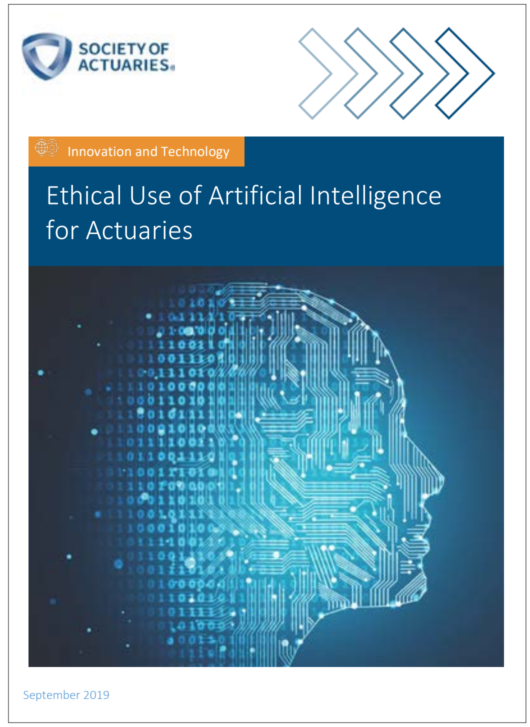
SOCIETY OF ACTUARIES
MODELING SECTION

ACTUARIAL TECHNOLOGY TODAY
JULY 2020

Ethical Issues in any Automated Decision-Making Model

By Neil Raden



Editor's note: The views expressed in this article are solely those of the author and do not necessarily represent the views of the Society of Actuaries.

The pace of new technology creates difficult ethical questions for insurance companies. The accelerating use of unattended decision-making applications opens the door to risk of reputation and liability. AI and Machine Learning (ML) models can contain bias, provoke discrimination, intrude on privacy, and unwittingly violate regulations. Unlike your other applications, there is no code to follow when something goes wrong; AI models are devilishly difficult to explain. Risk arises from the skill of the team, the quality of the data and issues of placing algorithms into production systems.

There are a multitude of risks from the development process itself such as: unclear goals, improper and unprepared data, data produced by unregulated third parties, overly enthusiastic desire to fit a model, and overconfidence when the first few "proofs of concept succeed," but plans to scale are incomplete.

In September 2019, the SOA released my research report, "Ethical Use of Artificial Intelligence for Actuaries," to cover the essential concepts of risk, how AI works, how to form an AI team for actuaries and other topics, such as:

- AI, the social context and the five pillars of ethical AI;
- digital transformation, future technologies and InsurTech;
- ethical risk in general: data, bias and do-It-yourself risk;
- ethical aspects of automation;
- organizing the actuarial department for: skills, team, diversity;
- IT, AI engineer and actuary team;
- government and regulatory initiatives;
- advanced analytics types;
- review boards: discussion and examples; and
- path forward.

We presented our findings at the SOA Annual Meeting in Toronto, in an SOA podcast, at the Enterprise Risk Management Symposium of the Casualty Actuarial Society in Tampa, Fla. (I was not able to attend, but Al Klein of Milliman artfully presented) and in an SOA webinar and newsletter in June.

The report is available to download from our website; we developed the subject matter for actuarial organizations but found a wider audience of insurance organizations such as general insurance, reinsurers, and insurance industry professional organizations, all of whom find the material current and relevant. We know that many of the issues in this paper will resonate with you.

ISSUES THAT DEMAND MORE DISCUSSION

AI is moving so fast, and new ethical issues are apparent. It is time to review the subject, first by commenting on what has materially changed in the last few years, and what ethical issues have arisen.

Specifically, the White House Office of Science and Technology Policy (OSTP) announced 10 Principles for Stewardship of AI Applications. These are important indications of the federal government's direction. Some states in the U.S. already put a stake the ground, not to mention the EU, and the Organization for Economic Co-operation and Development (OECD), an intergovernmental organization that represents 37 countries for economic concerns and world trade. OSTP probably shouldn't have bothered, as there was nothing new or even provocative in the 10 principles, and it has no teeth. If anything, it most resembled similar doctrines from China.

EIGHT AI ETHICS ISSUES TO CONFRONT

Ethical issue #1

How can organizations follow an ethical path with AI when the central government gives no guidance? The State of Washington just signed into law landmark legislation about facial recognition. According to the *WSJ*, "Washington state adopted a Microsoft Corp-backed law enshrining the most detailed regulations of facial recognition in the U.S., potentially serving as a model for other states as use of the technology grows." But should we entrust these issues to be addressed at a per-state level?

Ethical issue #2

Should a mega-tech company be writing legislation about a controversial AI application? Not everyone is comfortable with this, according to the *WSJ* article. Some feel the bill gives Washington State too much leeway. One provision allows police to use the technology without a warrant if "exigent circumstances exist."

We covered data bias in a report, "Ethical Uses for Artificial Intelligence for Actuaries," but it needs some explaining here. Data is an ethical problem, and always has been. Businesses should take every effort to minimize risks from data, especially when the data is from a third party, even *Data.gov* or *CDC.gov*, because data on its own has no context. How it was recorded and under what logic is missing when looking at a table.

There must be transparency around lineage, acquisition methods, and model assumptions, both initially and on an ongoing basis when the data is changing. There must be mandated security procedures to prevent loss from tampering and introduction of malware—all reinforced by comprehensive rights to audit, seek injunctive relief and terminate. The problem is that data brokers are mostly unwilling and unable to provide this.

Ethical issue #3

AI engineers, data scientists and predictive modelers crave new data. There is an aching desire to try all kinds of data to see if they can improve their models. The issue is that many data brokers are unscrupulous, and developers are wittingly or unwittingly poisoning their models with bad data. The problem is even worse when the data is reliable but the motivations of the modelers are less than pristine.

A good example is the use of credit scores for underwriting personal auto insurance. While there is an undeniable correlation between poor credit and risk, the causal relationship is not the same. Poor credit is a function of many societal factors, it is not the driver. The working poor are forced to carry car auto insurance which is expensive, a regressive tax, and if they fail to make a payment, they are likely to get a ticket or have their vehicle impounded for hundreds of dollars, depriving them of the right to have a vehicle to go to work, to transport their kids to school, to travel to a decent grocery store, all exacerbating their situation.

Some decisions cannot be made by matching against known patterns. According to Vegard Flovik in "How do You Teach Physics to Machine Learning Models":

"If there is no direct knowledge available about the behavior of a system it is not possible to formulate any mathematical model to describe it in order to make accurate predictions."

In "Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge," Emmanuel de Bezenac adds that the most prevalent use of AI (outside defense and intelligence, where it is not possible to gauge its breadth) is in targeted selling. The reason is in selling applications en mass produces the cost of being wrong sometimes is almost zero. If you process 100,000 credit requests a day and get 1,000 wrong, you still made 99,000 correctly. There are other kinds of decisions that require a higher percentage of correct responses. Little decisions add up—those judged unfairly will notice at some point in time.

Ethics issue #4

The people building AI are not sophisticated enough to engineer-in domain expertise. Here is the big potato: job loss from automation. One school of thought is that most jobs have unseen complexities that currently require a human in the loop, such as different types of data a machine can't cope with, or the person who remembers birthdays with thoughtful presents. This subtlety and finesse is never described. Automation can only go so far as the AI engineer understands the job.

In many periods of realignment, organizations found that staff made redundant were responsible for many tasks that were never recognized. AI obviously cannot replace things it isn't aware of. However, learning AI watches and learns, and as time goes on, more work is done by the machine than the person—and you get a mix of human agents and cognitive robots working together. But as time goes by, the proportion of work done by the robot could increase and the human part could decrease.

Ethical issue #5

Good intentions are that the AI will augment workers not replace them. This overlooks the learning aspect of AI, and organizations may fail to plan for the situation where the employee actually becomes redundant. And why is all this on the employer?

Because AI as a machine doesn't have an ethical framework; we have to give it an ethical framework.

If you put in enough data at the right level of quality, the AI will eventually become very good at spotting a pattern, and can tell you about it. That may be good for making recommendations to buyers, but for more multiplex problems, the question of what to do next is complex. AI cannot, at this point, tell you what to do next. The only way is through modeling and simulation. Data never speaks for itself. With ML, the action is not learned. It is predetermined: "If you see this pattern, perform this action."

Ethical issue #6

Understand the limits of what the AI can tell you. Conway's Law: organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations. In implementing AI solutions, developers must be aware that people are diverse and complex and live within groups and cultures. AI is not like coding. In many cases, there is no coding at all. There is nothing to examine for potential bias. As a result, data selection and labeling, feature engineering, model development and review all reflect the attitudes and belief of the group. No diversity in the group leads to insensitiveness to those affected.

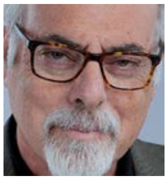
Ethical issue #7

It is too easy to be lulled into exposing personal information. In fact, it is too easy for bad actors to snatch personal data when you're not looking. Federated learning is a powerful idea for distributed applications and data, *https://ai.googleblog.com/2017/04/federated-learning-collaborative.html*, but first movers have chosen medical data as a testbed. Tread carefully.

Ethical issue #8

Using AI ethically ought to reflect that diversity is essential. "Fairness" isn't uniform; there are different versions of it. The emergence of Federated Learning, on the one hand, has positive implications for privacy, but on the other, is likely to exacerbate the explainability issue.

This is hardly a complete list—so it will be a recurring series. ■



Neil Raden is an active industry analyst, consultant, speaker and author of "Ethical Use of Artificial Intelligence for Actuaries." He can be contacted at *nraden@hiredbrains.com.*

Changing Your Analysis Mindset: Visualizing the Data

By Helen Wall

In today's world, the lines between technology and business areas are blurring into a less segmented and more integrated structure. Less than 10 years ago, IT would manage databases and produce business reports on this data through processes like SQL queries. The business would then consume these reports to analyze why they are encountering variances through their own tools like Microsoft Excel.

Today, however, we're beginning to see how businesses are leveraging applications that enable them to do much of the advanced analysis themselves and even dive into new analytics frontiers such as machine learning and AI. These new areas of explorations, in part, come from the increased volume and speed of today's data and a demand by the business to understand more about this data and make decisions with it. The newest applications enable a more efficient and flexible way to approach the process. Widely used platforms like Microsoft's Power BI and Tableau sit at the forefront of letting business users take control of how they interact and learn about the data on their own.

DIVING INTO THE PROCESS

While many people associate data tools with the beautiful visualization outputs illustrating the data trends, visualization represents just one segment of the entire data analytics process. Much like an iceberg where you only see about an eighth of it above the water, these data visualization applications require a lot of hard work and thinking on the backend to get them set up correctly.

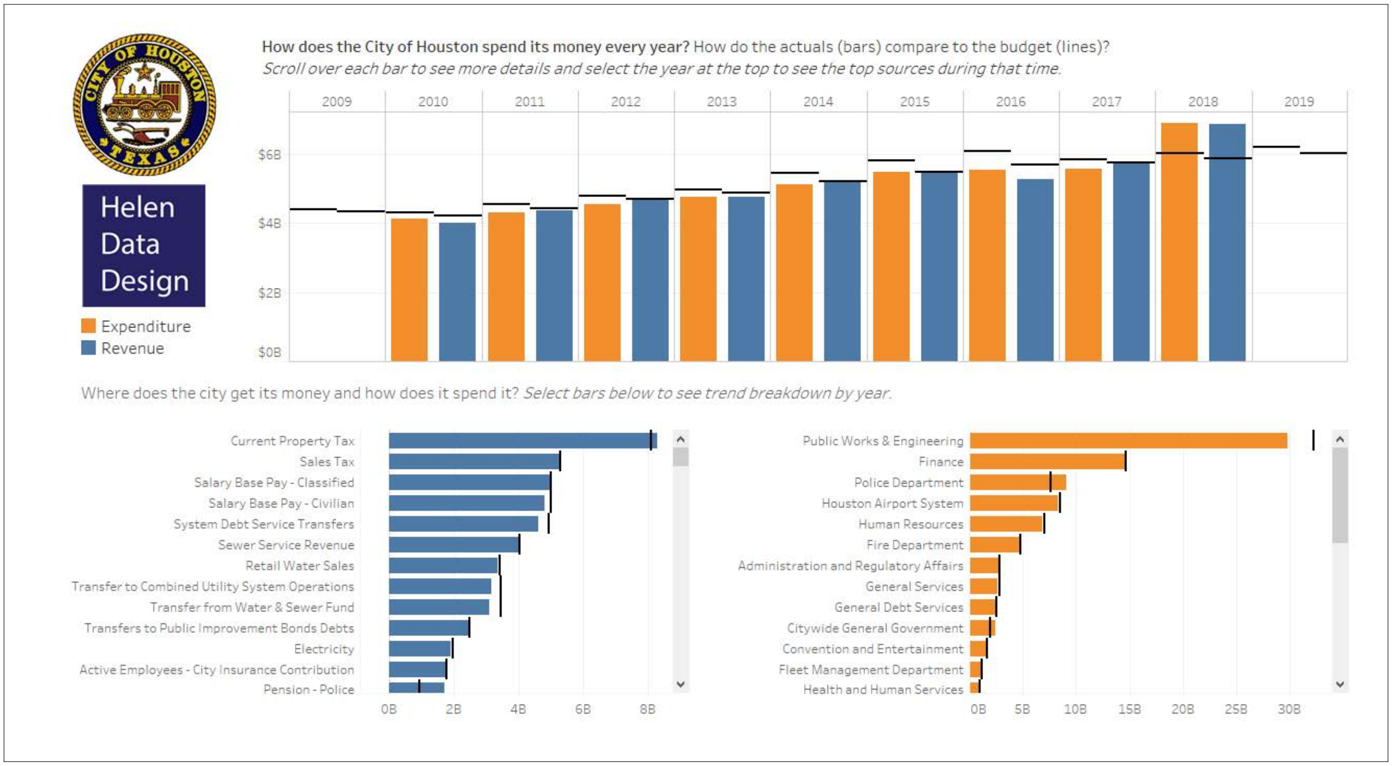
The process for creating a final data visualization dashboard goes like this:

1. Determine data sources and set up connections to them,
2. transform the connections into useful data tables,
3. join up tables to create a consolidated model,
4. create advanced calculations to measure and analyze the data trends,
5. design visualizations to communicate key trends and metrics, and
6. upload and share these scalable dashboards with the larger audience of business users.

Creating dashboards enables you to gain scalability and eliminate duplicated work. The key questions we are trying to answer is **why** something occurs and **how** the drivers affect the numbers.

To illustrate this, I will showcase an example dashboard. One of the most prevalent business intelligence tools in the industry market is Microsoft's Power BI, which combines the features of many applications like Excel and SQL Server Analysis Services, but also allows for an incredible amount of flexibility in creating models. You can really create any kind of financial model you want to within Power BI. However, a drawback is that you can only develop models on Windows and online, and sharing your work can be cumbersome if you don't have a paid subscription account.

Another intelligence tool is Tableau, which does have a public version of the desktop application, and has a public visualization gallery that enables us to explore data analytics with public data sources. While the modeling capabilities are more limited in Tableau, it does have beautiful visuals that serve us well for illustrating the impact of effective data visualizations. For my example, I'm going to analyze the financials for the city of Houston, where I live. They have a public portal to obtain the data that



you can explore on your own, or you can also search for datasets from many other municipalities on the internet.

DATA VISUALIZATION BEST PRACTICES

To get the data into a workable format to create visualizations, it often takes a bit of legwork to shape and transform the data. The data field names and values within the fields differed by year, and since I wanted to use a uniform set of keys to compare the data across a 10-year time span, I had to go through the data quite thoroughly to line up the segments by year. After that, it could create the keys to identify the segmented areas to examine these data trends across the entire time span. You can see the end results below. The visual that occupies the top part of the dashboard illustrates the revenue and expenses over the 10-year period. It also compares the actual amounts to the budgeted amounts by revenue and expenses each year. On the bottom left, you can see the comparison of revenue breakdown by area, and on the bottom right, you can see the comparison of the expense breakdown over the same time period.

While I could go through the steps undertaken to create these views, I think it's more important to highlight key approaches that you can implement in your own work to make it an effective communication tool. As you'll see, simplicity is often the best approach!

1. **Color theme:** I used blue for revenue and orange for expenditures consistently throughout this dashboard. You're not limited to these colors (I often incorporate gray into the color scheme as well). I am using a blue-orange-grey color scheme because it benefits colorblind users, who cannot typically differentiate between the common default red and green colors, illustrating value magnitudes on heat maps.

2. **Shape balance:** Bar shapes throughout the dashboard represent the actuals, while the floating lines hovering around the bars represent the budgets. This makes it easy to differentiate them but also eliminates too much noise going on in this view.

3. **Minimal visual count, but maximum effectiveness:** Utilizing more than three or four large visuals in a dashboard can cause it to become cluttered and difficult to read.

4. **Nudge prompts:** Right now, we see a stagnant view. How do we change what we see? Notice the prompts at the top to interact with the dashboard visuals. Think of them as subtle instructions that gently tell the user how to use the charts. I often refer to this as nudging prompts, and they're key to getting the appropriate amount of user engagement that you're aiming for when you design a dashboard.

5. **Put the elements within the chart in a sensible order:** Notice how I ranked the categories of both the revenue and expenditures, so it shows the highest numbers at the top of the visuals. This makes it easy for the dashboard consumers to identify the biggest drivers of the budgeted and actual numbers. The revenues versus expenses visual over time are in chronological order by year because this makes the most sense for time-series analysis.

6. **Remove duplicated or unnecessary labels:** Tableau initially sets up all these charts with more labels and components than what you see in the view. The key to being a good communicator through these visualization tools is to decide strategically what you need to include and what you can leave out to make your end communication of the data trends clearer and less cluttered for the viewer's eyes.

7. **Format tooltips:** One of the benefits of interactive visualizations are these third-dimensional components you can add to charts called tooltips. Don't just let the application automatically set up the tooltips and stick with its default displays. Try to experiment with creating your own views and key details about the data in these tooltips.

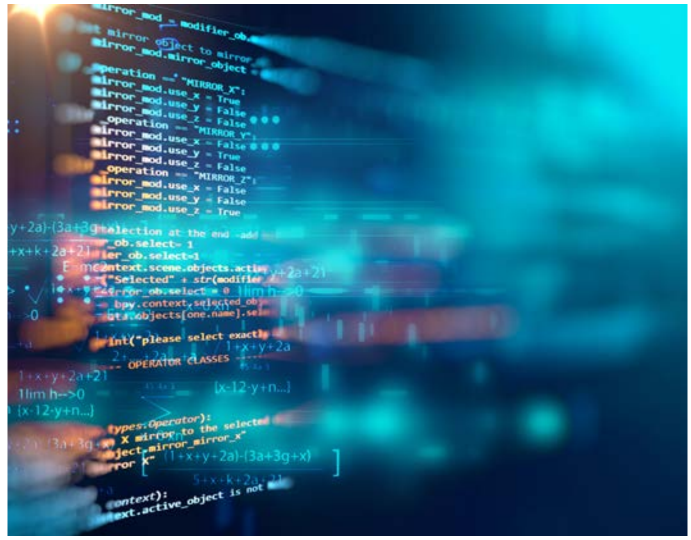
REFERENCES AND TUTORIALS

When the end-user interacts with the model, they can create their own view to analyze and understand the data. Try it out on your own in my Tableau public dashboard library! I spend most of my time exploring data analytics and creating data visualizations in Microsoft's Power BI. This City of Houston financial dashboard is a straightforward analysis of the consolidated dataset, but you can check out how to incorporate some neat models into these tools by experimenting on your own or following along with example projects. I recently wrote an article for *CODE Magazine* that walks through how to create an advanced financial model for life insurance calculations in Power BI. It walks through how to set up DAX measure calculations for both term life insurance premiums and reserves where the end-user can update the numbers directly by changing the input parameters in the dashboard.

There are many tutorials online that can offer a quick start to data visualization. For example, Power BI Data Methods focuses on the Power Query Editor, which also works in Excel! Power Query is one of the biggest things to happen to Excel in the last 20 years. Another example can be found at, Advanced Microsoft Power BI which gives a walk-through on how to create an interactive loan dashboard.

Happy data visualizing! ■

Helen Wall is the owner of Helen Data Design (*www.helendatadesign.com*). She can be contacted on LinkedIn *https://www.linkedin.com/in/helenrmwall/*.



Building a Modularized and Reusable Formula Table Code in Moody's Axis Using Formula Link

By Bryon Robidoux

When I ask actuaries about Moody's Axis, I get the impression that people think it's a black box system without the ability to customize pragmatically, but this couldn't be farther from the truth. Axis allows actuaries to customize its routines with VB.NET, which is a standard .NET Microsoft programming language. This article will be focused on maximizing the reuse of code using features available within Moody's Axis specifically targeted at using Formula Link.

AN INTRODUCTION TO MOODY'S AXIS

Non-Axis users may need a frame of reference for its two major components: Enterprise Link (E-Link) and the dataset. The dataset is the model, such as variable annuity or life insurance valuation model. E-Link has a very Windows Explorer feel. E-Link's main goal is to manage the collection of the organization's models and orchestrate their execution. E-Link can be automated with scripts to externally manipulate datasets and customize orchestration using Axis Jobs and E-Link scripts, respectively. For example, actuaries can write scripts to load in assumptions from a database with Axis Jobs for multiple datasets and then run each dataset's calculations using E-Link scripts.

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 datasets and E-Link scripts. 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's interface, it contains formula tables, code snippets, and other components which are not important for this article.

Formula Table Introduction

Within a formula table, the user can write specialized code. A formula table can be defined for many different calculation types. The calculation type will dictate the Axis specific variables and functions that are available for use in the custom code. A code snippet is a special formula table that can be used within any calculation type. The caveat is that it will not expose in the user interface what variables and functions are available because the variables and functions available will not be resolved and linked until runtime. This may seem like an issue or disadvantage, but actually, it is their greatest strength and gives them maximum reusability. It definitely makes them a little more challenging to use, though. Just note, the less specific the code is on what it does, the more versatile and reusable the code will be.

Now each formula table only supports Axis Script by default. Back in the day, Axis Script only supported VBA \ VB6 code. When formula tables were updated to support VB.NET, the precompiler was enhanced to force modelers to still code in the VB6 style to maintain backward compatibility for Axis's functionality. This evolution of formula tables has a major impact on their behavior because the Axis Script has different and much more restrictive syntax rules relative to VB.NET. As a person that has spent many years focused on improving coding methods in Axis, it is highly recommended to only use Axis Script for very simple products. As the complexity of the products increases, the more difficult it is to write clean and maintainable code. It is recommended for the organization to invest in the modeler's productivity and purchase Formula Link.

If the organization has upgraded to Formula Link, the formula table contains three tabs for code development: Formula Text, Functions and Variables, External Declarations and Classes. The Axis Script becomes the Formula Text tab after the upgrade. The Functions and Variables, External Declarations, and Classes tabs are far more compatible with VB.NET coding style. The three tabs do have different syntax rules, which lead to preferred coding behaviors that should be addressed.

It is encouraged to do most coding on the Functions and Variables, External Declarations and Classes tabs because they will force writing in functions and classes for better modularization. Only use Formula Text tab for calling functions created in the other two tabs and declaring constants because the Formula Text tab has heavy manipulation during pre-compilation. This manipulation makes writing clean code and using .NET language features very difficult or impossible. It is highly recommended to use Visual Studio as the debugger to immensely improve the modeler's productivity.

LOGIC PROLIFERATION AND CODE DIVERGENCE WITHIN THE DATASET

A common problem I have witnessed—which leads to logic proliferation and code divergence—is multiple formula tables having mostly identical code. (Code divergence is when different blocks of code should behave the same, but they don't because they are copies of each other and only a subset of the copies have been modified or enhanced.) To demonstrate, let's have two formula tables called FTA and FTB. Let's pretend that each formula table represents products A and B, respectively, and contains 51 lines of code each on the Formula Text tab. The first 23 and the last 23 lines are identical between both formula tables, but the middle five lines are slightly different for products A and B, which are displayed in Figure 1.

The Problem

Now let's go one step further and pretend that these functions were developed by one modeler that has moved onto another company. Shortly thereafter, a stakeholder reports an issue with a particular set of policies in product A. The new modeler determines the issue to be in the first 23 lines of FTA. She only changes FTA not realizing the redundancy or not wanting to cause an impact for products B. Maybe it was correct that the first 23 lines differ for product A and they should be changed. Maybe the code needs to be the same for A and B, so it is wrong only to change A. At best, the code is unclear. At worst, the modeler has just created an unintended model divergence that should not exist. The business requirement may be lost to the sands of time or not very clear itself. This is why it is important to write code that is very explicit on its intent. It should be written in a fashion that readers in the future can quickly assess what it's doing without having to find the original documentation. The code is not just for the compiler! It is also for the actuary to read.

The Option 1 Solution

How can development methods be improved to avoid this issue? The best way to prevent this situation is to create three code snippets called A, B, and Common. In each code snippet, A and B put a method called Calc on the Functions and Variables tab. They all must have the same signature.

"A function signature (or type signature, or method signature) defines input and output of functions or methods. A signature can include: parameters and their types, a return value, and type, exceptions that might be thrown or passed back."[1]

Then in code snippet A, copy and paste the five lines that specialized for FTA and save. Do the same process for code snippet B using FTB as displayed below. (Do not worry at this point if code snippets won't compile.) (See Fig. 2)

FIGURE 1
PRODUCTS A AND B CODE DIFFERENCES

```
'FTA
'Common init. for 23 lines
Const myArray() As Integer = {1,2,3,4}
Dim reserve as Double
        reserve=0
For Each item In myArray
reserve += 2 * item + 3
Next
'Common Summary 23 lines
```

```
'FTB
'Common init. for 23 lines
Const myArray() As Integer = {1,4,8,10}
Dim reserve as Double
        reserve=0
For Each item In myArray
reserve += 2.5*item+3.5
Next
'Common Summary 23 lines
```

FIGURE 2
CODE SNIPPETS A AND B

```
'Code Snippet A
Public Function Calc(ma() As Integer) As Double
        Dim reserve as Double=0
        For Each item In ma
                reserve += 2 * item + 3
        Next
        Return reserve
End Function
```

```
'Code Snippet B
Public Function Calc(ma() As Integer) As Double
        Dim Reserve as Double=0
        For Each item In ma
                reserve += 2.5 * item + 3.5
        Next
        Return reserve
End Function
```

In the Common code snippet on the Functions and Variables tab, as displayed below, create a sub routine called Common. Within the Common sub routine, move the first set of 23 lines from FTA, call the Calc function that has the same signature as the Calc functions that are in code snippets A and B, move the last set of 23 lines from FTA and save. (Do not worry that this will not compile at this point because it is missing the definition of Calc. It is all part of the plan.) (See Fig. 3)

FIGURE 3
CODE SNIPPET COMMON

```
'Code Snippet Common
Public Sub Common(myArray() As Integer)
        'Common init. for 23 lines
        Dim reserve As Double = Calc(myArray)
        'Common Summary 23 lines
End Sub
```

Next, delete all the previous code in FTA and FTB from all tabs and save because it will now be replaced. In the Functions and Variables tab of FTA and FTB, do what is displayed in Figure 4.

FIGURE 4
FORMULA FOR TABLE A AND B

```
'Formula Table A
IncludeScriptFromTable("A")
IncludeScriptFromTable("Common")
Public Sub CalcProd()
        Const myArray() As Integer= {1,2,3,4}
        Common(myArray)
End Sub
```

```
'Formula Table B
IncludeScriptFromTable("B")
IncludeScriptFromTable("Common")
Public Sub CalcProd()
        Const myArray() As Integer = {1,4,8,10}
        Common(myArray)
End Sub
```

Lastly, on the Formula Text tabs of FTA and FTB place the following function call:
FormulaTable.CalcProd()

The pre-compiler will merge the two snippets together at compile time, and each formula table will work as it originally did, and the redundancy is removed. (At this point, FTA and FTB should compile. If the user of the dataset wants to see the results of the merge, they will have to debug the code.) This is a really neat feature of Axis, which is typically not available in .NET. Anyone familiar will C++ will recognize this as a poor man's static polymorphism.

> "The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes, and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function."[2]

In this case, the type of the object that invokes the function is code snippet A or B.

Solution Option 2—Avoid Code Snippets and Use Classes Instead

Now let's imagine that the actuaries developing models are intimidated by the static polymorphism described above because they cannot look at the formula table and directly read the code without debugging. Is there another way to accomplish this level of reuse? Yes. The External Declarations and Classes tab allows users to create classes using dynamic polymorphism.

Hence, the actuary could create a code snippet called Common and write a class called Common on the External Declarations and Classes tab. It would contain a method called Calculate, which takes a parameter of an interface of type IProduct. The Calculate method comprises all the code from the Common code snippet's Calc method from Figure 3. (See Fig. 5)

FIGURE 5
COMMON CODE SNIPPET'S CALCULATE METHOD

```
Public Class Common
   Public Sub Calculate(theProduct As IProduct)
      'Common init. for 23 lines
      Dim reserve As Double = theProduct.Reserve()
      'Common Summary 23 lines
   End Sub
End Class
```

Within the same Common code snippet and External Declarations and Classes tab, create an interface IProduct that has one method called Reserve, which has the same signature as the functions in code snippets A and B. (An interface is a special class that contains methods that do not have any implementation. The reader can think of them as defining a set and its behavior. The classic example of an interface is the shape, which can have a method draw. Each implementation of an interface, such as circle and square, will define the specifics of how to draw it.) (See Fig. 6)

FIGURE 6
RESERVE SNIPPET

```
Public Interface IProduct
      Function Reserve() As Double
End Interface
```

In FTA, create a subclass A that implements IProduct on the External Declarations and Classes tab and do the same for FTB. Each subclass A and B contains the code that is in the Calc methods of code snippets A and B, respectively, mentioned above. The results of the transformation are displayed in Figure 7.

FIGURE 7
IMPLEMENTATION OF IPRODUCT

```
IncludeScriptFromTable("Common")
Public Class A
  Implements IProduct
     Private Dim myArray={1,2,3,4}
     Public Function Reserve() As Double _
            Implements IProduct.Reserve
                   Dim reserve as Double=0
                   For Each item In myArray
                     reserve += 2 * item + 3
                   Next
                   Return reserve
     End Function
End Class
```

```
IncludeScriptFromTable("Common")
       Public Class B
         Implements IProduct
       Private Dim myArray={1,4,8,10}
       Public Function Reserve() As Double _
              Implements IProduct.Reserve
                     Dim Reserve as Double=0
                     For Each item In myArray
                       reserve += 2.5*item + 3.5
                     Next
                     Return reserve
       End Function
End Class
```

In FTA on the Functions and Variables tab, create a method called CalcProd, which will instantiate a Common object and an A object and pass the A object to the Calculate method of the Common object. Do the same for FTB. This is displayed in Figure 8. (An instantiated class is called an object.)

FIGURE 8
CALCPROD METHOD

```
'Formula Table A
Public Sub CalcProd()
     Dim mediator = New Common()
     Dim prod = New A()
     mediator.Calculate(prod)
End Sub
```

```
'Formula Table B
Public Sub CalcProd()
       Dim mediator = New Common()
       Dim prod = New B()
       mediator.Calculate(prod)
End Sub
```

Now, to make the static and dynamic polymorphism examples equivalent, write the following line FTA and FTB on the Formula Text tabs.

FormulaTable.CalcProd()

Advanced Topic and Full Disclosure

The astute reader may have noticed that I put the IncludeScriptFromTable call in the External Declarations and Classes tabs of the formula tables. This was no accident. Remember when I mentioned in the introduction of formula tables that different tabs have different syntax rules due to the evolution of the formula tables and code snippets? The behavior that I expected is that the pre-compiler would line up the the tabs of the code snippets with the tabs of the formula tables and then merge the code snippet code with formula table code. This would ensure that all the code stays in its homogenous tab and gets compiled correctly, regardless of which tab the IncludeScriptFromTable call is made from in the formula table.

But instead, the precompiler looks at the tab that the IncludeScriptFromTable call was made from in the formula table. It then merges all the code snippet code into this one tab and compiles all the code. The code that was on a different tab in the code snippet versus the tab the IncludeScriptFromTable call in the formula table will fail to compile because an incompatible version of syntax rules will be applied. This is why the directions are very specific in the examples on the tab that the IncludeScriptFromTable call is located.

This has some undesired consequences because this means that multiple tabs cannot be used within the code snippets because one of the tabs would have compilers' errors because the syntax rules wouldn't match up. This means that if the common code grew and multiple tabs were needed to best express and abstract the concepts, they would have to be put into multiple code snippets. This really forces the actuary to make less readable code to overcome this issue, which makes the static polymorphism appear a little less slick. I am hoping that in the future, Moody's will change the behavior so that placement of the IncludeScriptFromTable call can be in any tab and not impact the rules of the compilation of the code within the code snippet.

OPTION 1 VS. OPTION 2

When should the actuary use static polymorphism versus dynamic polymorphism? The advantage of the dynamic polymorphism is that it is more transparent, which is always a good thing. There is still an issue, though. It can never be simple! The real determination of which method to use is the frequency at which the calculation will be called. If the code is called for every scenario, policy, time point, and whatever, then the system is allocating and deallocating memory at this frequency. This can be computationally expensive and time-consuming or can cause memory to become fragmented. These issues can be solved by declaring classes as static so that the instantiated classes keep their state between formula table calls. This causes another issue; the actuary is in the memory management business deciding when objects should be created and destroyed. Depending on the situation then, the run could crash due to a lack of memory. (For example, assume the Common class had a list that needed to be reset on a periodic, predictable basis. If a bug existed, the list might not be reset properly and grow unbounded.) The advantage of the static polymorphism is that the actuary can understand the polymorphism, but Axis is responsible for all the memory. Axis has the DIM_STATIC_VARIABLE to hold static between formula table calls to replace the need for the list used in the dynamic polymorphism example. Memory management is very difficult to implement correctly, so it is best to delegate it away to Axis because the platform is focused on the problem. The difficulty of memory is why static polymorphism is preferred, and C++ lost its popularity.

RESULTS OF REFACTORING

This pattern of separating common from specialized code can be repeated over and over again. It is highly recommended to avoid coding directly in formula tables so that the above pattern is encouraged. It has many advantages:

1. It clearly defines the parts of the algorithm that are common.
2. It specifies exactly where code variations occur.
3. If the algorithm is wrong in the common code, it can be changed in one location and fix everything at once.
4. It allows the code to be compressed as much as possible.
5. It allows the code to be divided into smaller and smaller pieces for better maintenance and comprehension.
6. It can be easily extended for a future product C, and so on, by creating the code snippet and following the pattern.

LOGIC PROLIFERATION AND CODE DIVERGENCE AMONG THE DATASETS

The redundancy and logic proliferation might be caught within one model, but now imagine the variations exist in different models. There is no native tool from within the dataset that can overcome it. Luckily in September 2019, Formula Link code snippets were introduced to save the day! The only code that would change from handling redundancy **within** a dataset versus **among** datasets is to change line 4 from IncludeScriptFromTable("Common") to IncludeScriptFromFormulaLinkTable("Common") in each of the FTA and FTB formula tables displayed above. Lastly, the code snippet would have been removed from the dataset and saved in Formula Link.

This way the user can keep the common code in a centralized location that is visible to all models. The unique variations of each model are stored in the dataset and injected into the code at compile time. For example, imagine that the common code was an economic scenario generator (ESG) that both a Fixed Indexed Annuity and Variable Annuity dataset would need for projecting liability values. They could keep the common code of interacting with the ESG in Formula Link and keep all the specifics of how the liability needed to interact with it inside the dataset. Code snippets in Formula Link give the modeler the ability to avoid copying regardless of where the redundancy exists—which is exactly what good software engineering principles dictate.

Now for Transparency

Now that the beauty of Formula Link was addressed with code snippets, the difficult side of using Formula Link needs to be exposed. The "Formula" in Formula Link comes from the ability to write code outside the dataset. The purpose is to allow the user to:

1. Write reusable Dynamic Link Libraries (DLL) within the Moody's environment using object-oriented C# or VB.NET classes; and

2. to link-in external libraries' DLLs that were written outside of Axis.

Bullet 2 is an awesome feature, and its potential will be shown in a future Modeling Section article, "The Importance of Centralization of Actuarial Modeling Functions – Part 4 DevOps and Automated Model Governance." Bullet 1 is where the difficulties arise.

The difficultly with Formula Link has to do with using the Formula Link library classes directly. Formula Link library classes cannot directly call the functions or variables inside the dataset. There is no library that can be referenced to expose them. (This limitation is for justified technical reasons beyond the scope of this article.) In order to get a hold of the internal dataset functions and variables, the developer has to pass them to the Formula Link library classes directly. Passing functions requires using function pointers and lambda expressions, which are advanced programming skills. The library gets cumbersome and difficult to understand if it requires tons of parameters, especially functions as parameters. This is why it is highly recommended to use Formula Link code snippets over calling the Formula Link classes directly. When the formula table in the dataset calls the Formula Link code snippet(s), the dataset's pre-compiler will link all the datasets functions and resolve the dependencies. Following this rule of thumb will greatly reduce the complexity of the code and increase its readability.

The last suggestion is to set the dataset and Formula Link to Option Strict, which shuts off the ability to do implicit type conversion. This feature is especially important when using Formula Link because the types in the Formula Link library are not resolved until runtime. Hence, the dataset would compile and start a run, but possibly stop running due to a type mismatch error. The Option Strict will prevent this from happening because it will find the type mismatch during compilation. The directions on how to set Option Strict can be found in the knowledge base.

CONCLUSION

In conclusion, this article focused on code reusability and modularization by using code snippets. The art of coding is to be able to encapsulate the changes between similar concepts and then inject the variations. The injection of differences is accomplished through polymorphism, of which there are two types: static polymorphism or dynamic polymorphism. The static polymorphism is the modeler's only option without Formula Link. (The reason that static polymorphism wasn't shown using Axis Script is that Axis Script is so verbose.) The preference was on using static polymorphism because dynamic polymorphism can be computationally expensive and tricky to implement. Regardless of which method is used, it is important to write clear code so that future developers can understand its intent and therefore reduce confusion. The methods shown will help reduce redundancy of code in the model and make it easier to maintain. ■

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 *https://developer.mozilla.org/en-US/docs/Glossary/Signature/Function*

2 *https://www.tutorialspoint.com/cplusplus/cpp_polymorphism.htm*

Technology Section Tidbits

Stay engaged with your community

SECTION COMMUNITY

Have you thought about the future of the actuarial profession and how future technology will revolutionize it? Take part in Disruptions to the Actuarial Profession Contest to showcase your ideas about how actuaries can contribute to this process. To participate, submit a video, YouTube hyperlink or essay to ***aof@soa.org*** no later than Aug. 3. Up to five prizes will be awarded for ideas including most reformative, most creative and most innovative uses of technology. See our **contest flyer** for more information on how to enter.

PROFESSIONAL DEVELOPMENT

Take part in the **Tech Webcast Series—Emerging Tools & Technologies** which consists of four webcasts being presented on Wednesday's from July 28 through Aug. 18. This webcast series kicks-off with an introduction and use-cases relevant to the emerging tools and technologies. The subsequent webcasts take a deeper dive into solutions, considerations, and use-cases for each topic including cloud computing, data transformation and warehousing, and data visualization. Sign up for the entire series by Aug. 16 to take advantage of the special discount.

Get access to more info at ***SOA.org/sections/technology***