What's a Model? A Framework for Describing and Managing Models

By Dodzi Attimu

The definition of a model is one of the first items addressed in any model risk management program. Thankfully, the Federal Reserve's Office of the Comptroller of Currency's guidance in "SR Letter 11-7" on model risk management[1] provides a good benchmark (if not the standard) regarding what a model is for the financial industry. In this article, we will address some model-related questions that arise in operationalizing a model risk management program. Some of these questions may be philosophical while others are operational. One such philosophical question is whether a model is a *process* or a *unit* that transforms input via computational methods into useful output/estimates. An example of an operational question is whether to classify modeling functionality on a single platform (e.g., Prophet, MoSeS, GGY AXIS, etc.) as a single model or as multiple models.

Regarding the operational situation, a typical scenario is the following: An ALM projection functionality is built on a platform like GGY AXIS for insurance products Product_1 and Product_2 that generates cash flow (CF) projections that are used for Actuarial Guideline (AG) 43 and C3-Phase 2 analysis and reporting. The question becomes: Does this represent one model or four models (the latter corresponding to a model each for the two products times two business processes) or maybe two or three models?

In this article, we outline a *formal framework*[2] *for describing models* that is inspired by the operational context of governance, management and use of models. This is a coherent and consistent frame of reference to answer relevant questions related to models and their use. The framework also provides a sound and easy mechanism and "language" to articulate and analyze different design approaches for models that may have big impacts on the efficiency of business processes relying on them.

Another helpful feature of this framework is that it leverages the actual operational aspects of the use and maintenance of models. Consequently, we expect the framework and related ideas presented here to be of interest to model developers, model testers and model validators, individuals in model governance or model audit functions, as well as business users of models.

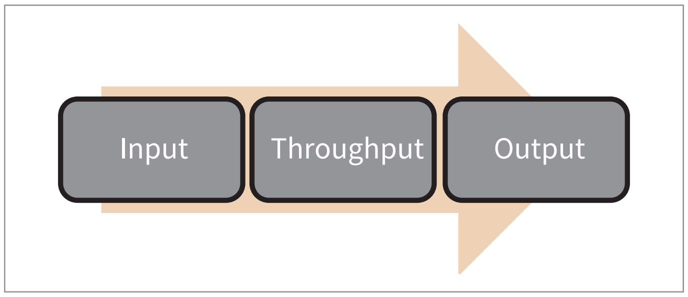
OK, SO HOW DOES ONE DEFINE A MODEL AGAIN?

The Fed guidance in SR Letter 11-7 on model risk management states: "… a *model* refers to a quantitative method, system, or approach that applies statistical, economic, financial, or mathematical theories, techniques, and assumptions to process input data into quantitative estimates."[3]

More pithily, a model is a means to transform input (data, assumptions and other parameters) via a processing component (throughput) into quantitative estimates (output).

Figure 1 provides a depiction of a model showing the constituent parts.

Figure 1
Depiction of a Model



Typically, a model is defined in the appropriate policy (or policies) of a model risk management program and is usually the exact definition in SR Letter 11-7 noted previously or modified based on the operational needs or priorities of the model risk management program. For example, in some programs, any tool that performs any sort of quantitative transformation is classified as a model, whereas other programs add an extra requirement that the transformations involve uncertainty or some element of judgment/assumptions.

In some instances, too, a model is defined as an end-to-end modeling process spanning the sourcing of inputs through extraction, transformation and loading (ETL), model run, and reporting based on the output. We would argue that this latter definition is of a business process that utilizes a model(s) because defining a model as a process provides both philosophical and semantic challenges.[4]

In this formal framework (which is based on the earlier SR Letter 11-7 definition), a model is not a process but an operational unit and is different from business processes that utilize it. Conversely, a business process would be one that may utilize models as tools. Thus, a *model validation* may focus mainly (but not exclusively) on this operational unit, whereas a *business/*

modeling process validation would include validation of all relevant components of the process, including the model(s) used therein.

Formal Structural Definition of a Model

Conceptualizing a model as an operational unit consisting of an input structure, processing component (throughput) and output structure is fairly high level in the sense that we do not go into details about what requirements the processing unit should perform to qualify as a model. That can be determined by a model risk management program as needed. The use of *operational unit* in this framework means that a model could use a collection of software libraries, input file structures and so on. However, if to use the model, the input structure (potentially a collection of structures physically represented by different files) is operated on collaboratively by these components, all these (potentially stand-alone) components are part of a single model.

Defining "Model"

A model is an *operational unit* consisting of an *input structure* and a *throughput* (processing logic/functionality) that acts on the input structure to produce an *output structure*. We will refer to the input structure, throughput and output structure as *operational components* of a model.

Sometimes, confusion arises regarding the use of the term "model" because it can mean a quantitative abstraction of reality (i.e., in the phrase "asset/interest rate/mathematical models") on one hand or an operational unit (what the definition of SR Letter 11-7 addresses) on the other. Consequently, a model (an operational unit) for valuing a portfolio of assets could consist of many different asset/interest rate models (abstractions of the value of these assets/risk factors). Examples include LIBOR Market Model, Black Scholes model, SABR model and so on.

In other words, if a code base (logical specification) encompasses several de facto "mathematical/financial models" but supports an input structure to generate an output structure, we *operationally* have *one model*. Without a framework for defining models from an operational standpoint, there could be (unnecessary) disagreement on what constitutes a single model or multiple models.

Sometimes, confusion arises regarding the use of the term "model" because it can mean a quantitative abstraction of reality on one hand or an operational unit on the other.

In addition, we note that the processing logic of a model consists of all logic that is accessible (reachable) through a unique logical entry point. At a high level, consider this entry point to be synonymous with a "RUN" button or a command that triggers the calculations.[5] The next sections expand on the three operational components of a model.

Input Structure

The first operational component of a model is the input structure. This includes user interface, configuration files and an input data structure that may reside in external files (which may be referenced from the user interface).

First, in this formalism, input includes raw data, assumptions and parameters. However, it is possible due to convincing reasons or just bad design that certain aspects of these are "hard-coded" in a given model's implementation code or set-up, that is, throughput.

Second, the use of *input structure* instead of *input* is because input consists of structure and content/values *leading to a distinction between input structure and input content/values*, which is a deliberate and important distinction in this framework. Input structure is the general "shape"/data structure of the inputs—for example, what type of inputs are expected and how the various elements are arranged—whereas input content refers to specific values for these inputs in the structure. This (input) structure can be represented abstractly as a collection of tables. Note that this choice is for convenience.

One natural motivation for the distinction between content and structure is for the purposes of defining what a model change is in this framework (see "Defining 'Model Change'"). In particular, we naturally have a situation where a unique model can have different input content and hence generate different output. In other words, changing an input value to a model does not result in a different model; it would be the same model producing different output.

Third, though our choice of tables as building blocks of input structure is for convenience, it is also general enough to support any form of input structure. This readily follows from the fact that for any set of inputs, one can construct a one-field table for each element of the input. One immediate outcome of this observation is that the representation of an input structure is not unique (e.g., one could organize the structure into two or three or more tables). Hence for a given set of inputs, different input structures could be used to support them. These structures could be different but capture the same information. Physically, the input structure could consist solely of one or multiple types of the following: relational database, text files, Excel files, special file format native to system,[6] and so on.

Finally, note that we consider the input structure as encompassing all that a collection of code, plus any other processing component constituting the model, operationally supports to process input values.

Throughput

The throughput is the second operational component of a model and refers to logic that transforms the input to provide estimates or output. In addition, we also consider as part of the throughput any other component of the model that is not considered as part of input or output structure. In other words, we include parts of modeling system that are responsible for generating and formatting output and performing modeling housekeeping activities such as validation of inputs as well; not just the business logic.

Output Structure

Similar to our highlighting of *structure* for inputs, we emphasize the structural aspects of the output. The output depends on the throughput. In addition, similar to the case of the input (structure), we will assume without loss of generality that the output has a structure of a collection of tables. Again, similar to the input structure, we consider the output structure as the union of all (table) structures supported by the code base via its point of entry.

Interrelationship Between the Three "Puts": Input Structure, Throughput and Output Structure

We first note that in this framework, software code that implements some logic, but has no functionality to provide output, does not qualify as a model. All three operational components must be present for a classification as a model.

Another aspect of the interrelationship between the operational components of a model is related to whether an assumption is part of the input or part of the methodology (throughput) of a model. For example, consider a (toy) model that projects stock prices under the Geometric Brownian Motion (GBM). The functional specification of this model is an assumption.[7] It is also correct to say that the volatility parameter is an assumption.

The challenge when talking about assumptions related to a model (e.g., appropriateness for a given modeling use) is to determine whether one considers the GBM specification as an assumption or only the volatility parameter. This is the motivation for the following two definitions.

Defining "Assumption Input"

An assumption that can be captured via the input structure is called an *assumption input*. Since input consists of both structure and content as noted earlier, we consider an assumption input as consisting of an *assumption input structure* as well as *assumption input content*.

Defining "Assumption Throughput/Implementation"

An assumption that is part of the implementation software code (processing logic/throughput) is called an *assumption implementation*.

Let us revisit the point earlier about different input contents to the same model in the light of our GBM asset projection system. Assume our input structure consists of four entries per stock: the number of time steps, length of time step, number of paths, and the volatility. Changing any of these input values does not result in a different model. An equivalent deduction is that the input content, while necessary to produce output content of a model, is not a component of the model. This makes the definition of a model in the framework an operational abstraction.

SOME APPLICATIONS OF FRAMEWORK

In this section we outline some applications of this framework. First, we answer the question of whether an assumption is part of a model or not. Next, we tackle the problem of determining if a component is part of a model. We then address the issue of determining the number of models represented under a given modeling setup for different products supporting different business processes/metrics. Finally, we consider in general some model management concepts of model design, change management and related activities.

When is an Assumption an Input or Part of a Model?

For example, consider the earlier simple model that projects stock prices under the GBM. Assume that the input structure supports a single (constant volatility) parameter (ignoring other input values supported by the input structure) per stock. Are the volatility parameters and GBM assumptions part of the model?

To answer this question, note that the volatility parameter value is an assumption input content and hence is not part of the model. In addition, recalling that input consists of input structure and input content, and that it is the input structure that is a constituent operational component of a model, we will say the *assumption input structure* is part of the model (though the *assumption input content* is not, as noted earlier). For readers who might struggle with the latter point, note that intuitively, one can change the content of the volatility input structure without creating a different model as a result (more on this in the upcoming formal definition of a model change). On the other hand, as the formulaic implementation of the GBM is fixed in the model processing code/logic, it is an *assumption implementation*; and since the throughput is an operational component of the model, it is part of the model.

However, another design could involve an input structure that captures the type of stock price evolution assumption specification as well as the parameters for the specification. For example, the user can specify either GBM or GBM with Jumps as input content/values in addition to the assumed input content/values of the parameters. In this case, the assumed functional specifications as well as parameters are part of the input, and we would conclude that the actual *choice of volatility input* and *model specification* used for the model is not part of the model—they are mere inputs into the model. This last design illustrates an important consideration for designing models that are flexible and operationally efficient. We hope to follow up with an article on elegant, efficient and flexible model designs with emphasis on user configuration of third-party projection platforms.

Determining Constituent Parts of a Model

Consider a vendor modeling platform[8] that has an operational unit used for ALM projection and has:

- Input structure supporting inputs like economic scenario input (projected yield curve for Treasurys and spreads over Treasurys) for all fixed income assets in the portfolio of assets backing general account liabilities, equity and dividend growth rates of indices mapped to separate account values (AV), liability policy data (AV, guarantee bases, age of policyholder, etc.), assumptions input (parameters for lapse formulae; GA reinvestment strategy, e.g., target allocation, reinvestment frequency, etc.) among others.

- Suppose also that this model projects the assets and liabilities and produces (via the throughput component) an output structure housing cash flows (assets and liability cash flows) and financials on a STAT basis by scenario and for each monthly time step for 40 years.

- The output structure consisting of at least one table with a field that captures scenario number and houses the monthly income statement output for 40 years.

  - At least one table because there may be other, lower-level information that constitutes the output structure, for example, debugging information that has intermediate calculations or calculation results at a lower level of granularity. These may be optional output that is supported by the throughput and hence is part of the output structure. For our purposes, it is the financial statement component of the output structure that is important in this example.

- After the results are generated in the output structure, an Excel Analytics tool calculates a conditional tail expectation (CTE) number among other analytics and graphs.

In this scenario, is the Excel Analytics tool part of the model?

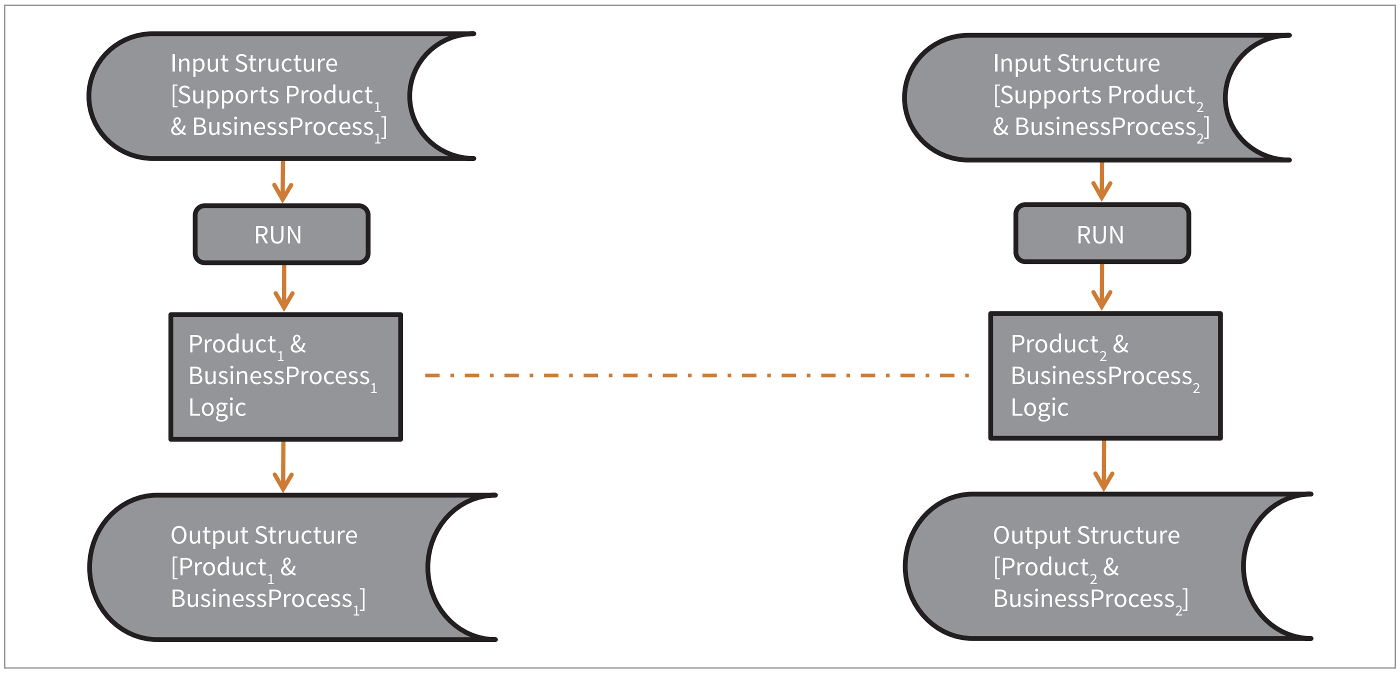
The framework gives a natural answer, which is it depends on whether that analytics functionality is part of the platform's processing logic (i.e., throughput). The reasoning follows naturally from throughput consisting of all logic that is reachable from the model run entry point. Consequently, in this example, if the analytics tool is a stand-alone tool that can only be activated by manually opening and using it without it being driven by the model throughput (via its entry point), it is not part of the model.

The natural follow-up is whether the Excel-based analytics tool can be made a de facto part of the model in this formal framework, and it can. (So fans of "model-as-a-process" paradigm can still operationally design a single model that touches all applicable processes.) To do that, it suffices to incorporate the analytics tool as part of the throughput. Operationally, one option is to add to the throughput some logic/functionality that triggers the working of the analytics tool directly in a way that is reachable from the entry point.

In other words, hitting the proverbial "RUN" button would run the model and trigger the analytics tool functionality. This does not have to involve removing the option to use the analytics tool independently on a stand-alone basis. The formal principle that is applicable is the so-called *enclosing/encompassing property of throughput*. This property posits that any *customized* (potentially independent/stand-alone) functionality (code, .dll, .exe, etc.) that is reachable (e.g., called) from the entry point of the throughput is a de facto part of the throughput.

This property is not just a purely abstract algebraic concept for its own sake.[9] Its importance derives from the fact that the operational model component of throughput usually consists of various (potentially independent, multi-technological)

Figure 2
Determining Number of Models


Input Structure [Supports Product$_1$ & BusinessProcess$_1$]
RUN
Product$_1$ & BusinessProcess$_1$ Logic
Output Structure [Product$_1$ & BusinessProcess$_1$]
Input Structure [Supports Product$_2$ & BusinessProcess$_2$]
RUN
Product$_2$ & BusinessProcess$_2$ Logic
Output Structure [Product$_2$ & BusinessProcess$_2$]

operational components. In addition, many customizations by users may involve adding .dlls, executables or scripts that add various functionality, including input validation, and analytics to the throughput.

Determining "Number of Models"

In this example we consider the case of a projection system that supports the projection of variable annuity liabilities for n products Product$_1$,...,Product$_n$. These products are all modeled on the same platform, say PlatformX. In addition, using this platform, m business processes (or metrics) BusinessProcess$_1$,...,BusinessProcess$_m$ are supported.[10] How many models are represented in this scenario? On one extreme, we have n × m models (one model for each product, business process (metric) combination). On the other extreme, we have one model that supports all products and metrics.

But what is the right answer? We show how to make this determination naturally (without resorting to subjective "judgment") using this formal model description framework. Indeed, the number of models in this case is determined by how many stand-alone operational units are represented in the modeling setup. In other words, it is determined by the *design/configuration of the model(s) on the platform*.

Let's delve deeper and show how to make the determination. Based on the formal definition, if there are n × m different operational units (consisting input structure, throughput and output structure) then there are n × m models. Without loss of generality, let's consider that there are two products, Product$_1$ and Product$_2$, and business purposes, BusinessProcess$_1$ and BusinessProcess$_2$. In one extreme, we could have four models (operational units) with the following model representation for each combination:

- A model for Product$_1$ and BusinessProcess$_1$
- A model for Product$_1$ and BusinessProcess$_2$
- A model for Product$_2$ and BusinessProcess$_1$
- A model for Product$_2$ and BusinessProcess$_2$

Diagrammatically, we illustrate any one of the four models in Figure 2. (We have shown the first and fourth models.)

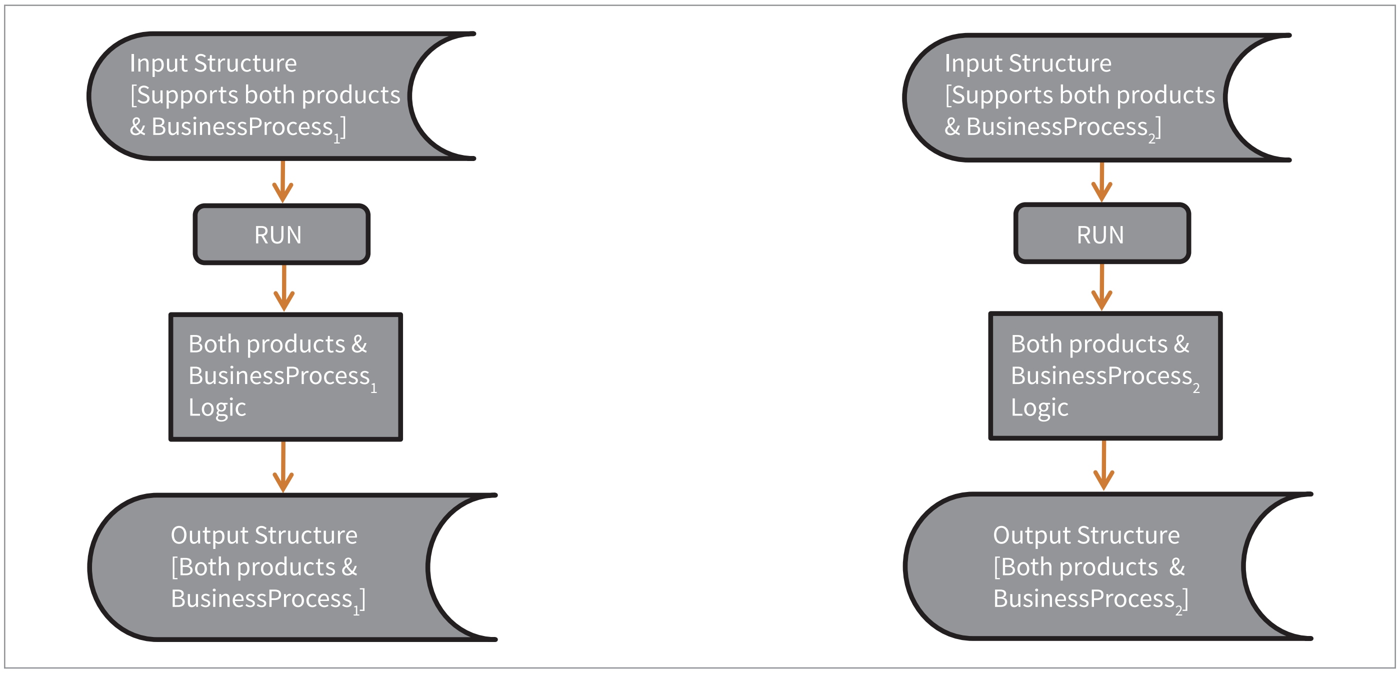
Another possible design would be a two-model design:

- A model for Product$_1$ & Product$_2$ and BusinessProcess$_1$
- A model for Product$_1$ & Product$_2$ and BusinessProcess$_2$

These are illustrated in Figure 3.

One the other hand, we could also design a single model to cover the products and business processes. To do that, it suffices to combine the processing components and hence the input (and output) structures. Without loss of generality, one need

Figure 3
Two-Model Design


Input Structure [Supports both products & BusinessProcess$_1$]
RUN
Both products & BusinessProcess$_1$ Logic
Output Structure [Both products & BusinessProcess$_1$]
Input Structure [Supports both products & BusinessProcess$_2$]
RUN
Both products & BusinessProcess$_2$ Logic
Output Structure [Both products & BusinessProcess$_2$]

only consider addition of fields that specify the product type and business process as part of the input structure. Naturally, this leads to a combined input structure that supports both products and business processes.

Note that this naturally satisfies the conditions for having a single model:

1. There is a single input structure that supports both products.[11]
2. There is a single processing unit that acts on the same input structure (that supports input contents representing both products and business processes).[12] Another way of saying this is that there is one processing code base for both products and business processes (metrics).
3. Typically, once the first two are satisfied, we have the same output structure for both products.[13] In other words, the input structure plus the throughput determines the output structure as well.

Figure 4 illustrates such a design.

A similar scenario is this: Given an asset modeling platform that supports the modeling of different asset classes $A_1,\ldots,A_m$, does this represent m models or some n < m models? Using similar reasoning as before, if the underlying code framework supports all the different asset classes, then this constitutes one model. If, on the other hand, there are stand-alone code (base) units that support the individual assets and these units are not reachable via a single entry point, then they can only be run as individual units and each such individual unit is a separate model.

Finally, a top-down mechanism for determining if the setup represents a single model consists of answering the question: "Can one utilize the same input structure and singular entry point ("RUN" command) to generate results for both products and business processes?"

Model Design Implications for Model Governance and Control

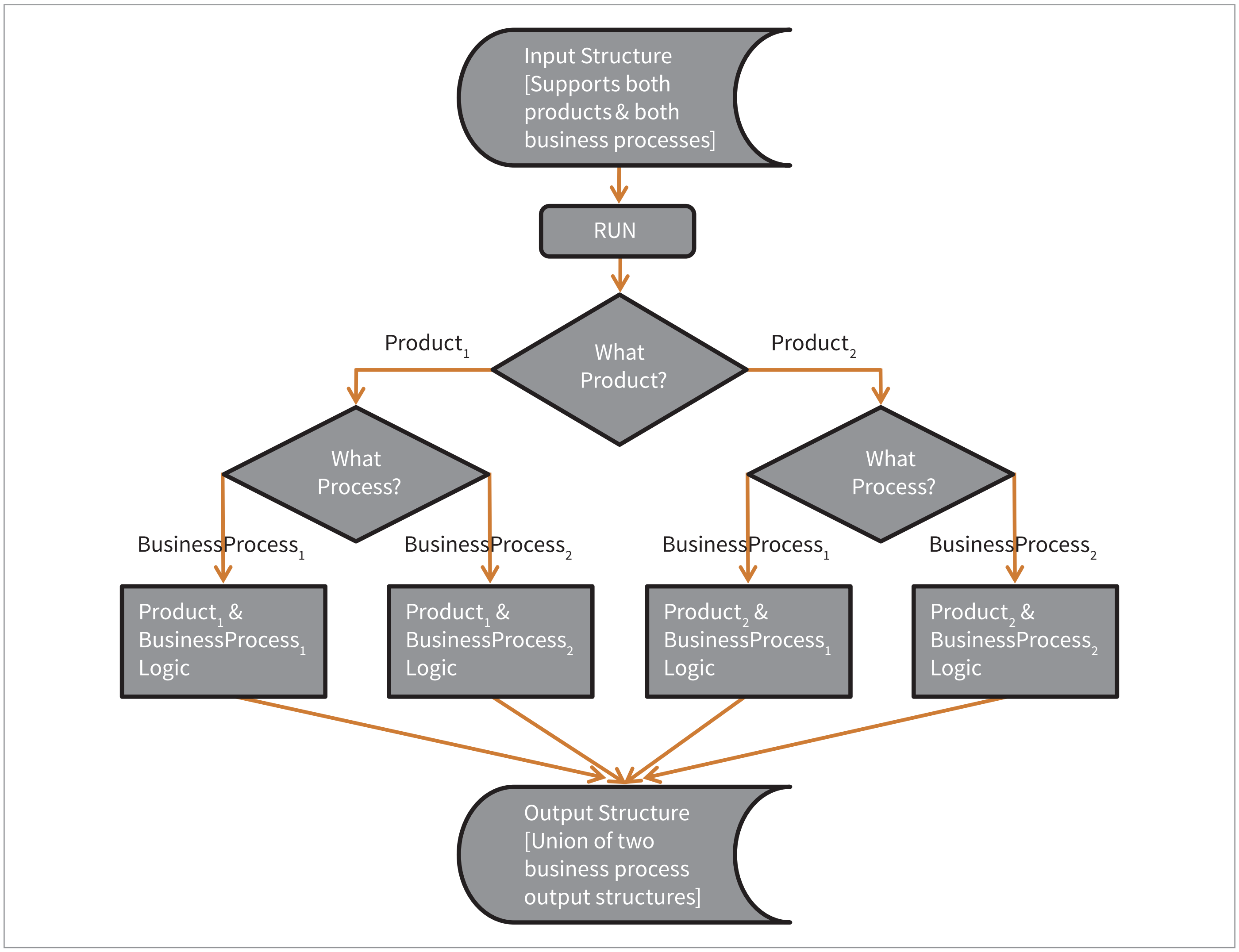
For a better appreciation of the implications for model design and controls, we propose the following definition of what a model change is using this framework.

Defining "Model Change"

A model change is defined as a change in either the input structure or throughput.

Now let's proceed by considering the example in the previous subsection. There would be one model that supports multiple processes if the model is designed such that its throughput can interact with an input structure that:

Figure 4
Single-Model Design


Input Structure [Supports both products & both business processes]
RUN
What Product?
Product$_1$
Product$_2$
What Process?
What Process?
BusinessProcess$_1$
BusinessProcess$_2$
BusinessProcess$_1$
BusinessProcess$_2$
Product$_1$ & BusinessProcess$_1$ Logic
Product$_1$ & BusinessProcess$_2$ Logic
Product$_2$ & BusinessProcess$_1$ Logic
Product$_2$ & BusinessProcess$_2$ Logic
Output Structure [Union of two business process output structures]

- Supports the modeling of different products; that is, the input structure supports different products

- Supports different types of business processes; that is, the input structure supports different business processes (e.g., AG43 output, C3-Phase 1 output)

In addition, as more assumptions and parameters are supported by the input structure (as assumption inputs), we have a situation where such assumption updates do not go through model change processes since they are input (content) changes, not model changes. This is a natural corollary of the definition and is consistent with one's intuition regarding "inputs."

The alternative is to make a determination of what constitutes a model change based on (subjective) judgment sans a framework. Interestingly, changing an input structure (as minor as that may be) is a model change, whereas changing an assumption such as the target allocation of a reinvestment strategy that could have major impacts on model results is not a model change if it is solely effected through the same input structure.

This does not imply that business users should not test and assess the validity of results coming from the model before putting the new assumptions into "production." It only means this is work that is outside *model change control* and is rather an *assumptions change control* process that can happen without triggering model change protocols.

There is a higher initial cost in setting up and testing models in this way, however. For example, test strategies should cover different input choice combinations to ensure that the abstraction(s) inherent in the model design are valid.[14] On the other hand, the advantages in flexibility, maintainability and efficiency of this input-driven approach can be huge. Users have more flexibility to perform analysis or assumptions updates that conform to input structure on the (official, validated and tested) model without triggering a model change process.

In certain cases, models are designed with inadequate consideration for what should be hard-coded vs. what should be part of input structure. This leads to duplication of models that are logically the same except for a few differences in the throughput. Over time, because these "similar" models develop a life of their own, they tend to diverge in unintended ways leading to potential problems and inconsistencies down the line.

In fact, the considerations for good model design are enough for an entire article, and we end this subsection by noting that a lot of efficiencies can be derived by shifting stuff that is traditionally considered as throughput into the input structure. Finally, using this framework to design elegant and efficient models is not only possible with home-grown systems but also with models built on vendor-supported software platforms like GGY AXIS, MoSeS and Prophet. As noted earlier, we hope to pursue this in a follow-up article.

CONCLUSION

In this article we introduced a formal framework for representing a model that is consistent with the financial industry standard definition of a model as seen in SR Letter 11-7. This framework operationalizes the definition of a model and naturally answers questions such as what functionality constitutes a model or how many models are represented by different projection capabilities or business processes. Finally, this framework also provides a natural and succinct way of communicating model design and hence improvements in existing designs or entirely new ones for modeling capabilities. ■

Dodzi Attimu, FSA, CERA, CFA, MAAA, Ph.D., heads the model validation program at MassMutual. He can be reached at *dattimu06@massmutual.com*.

ENDNOTES

1 Board of Governors of the Federal Reserve System Office of the Comptroller of the Currency, Supervisory Guidance on Model Risk Management (SR Letter 11-7), April 2011, *https://www.federalreserve.gov/supervisionreg/srletters/sr1107a1.pdf*.

2 The exposition in this article is a synthesis of concepts from the rigorous (algebraic) development of the framework by the author.

3 *Supra*, note 1.

4 The main philosophical/semantic challenge is that a model and a process are not necessarily interchangeable. For instance, consider that a model is a tangible thing (operationally), whereas a process is not; it is a sequence of operational steps, some of which may involve running the (tangible) model(s).

5 This concept is easier to grasp in models developed on programming platforms but requires more work to formalize in purely spreadsheet-calculation-based models.

6 We strongly believe that third-party platforms should be able to communicate with well-known file formats. They could convert data to some underlying native file formats as needed, but forcing the user to convert to native file formats is a bad use of users' time.

7 In other words, the stock price will evolve under the specification $S_{t+\Delta t} = S_t e^{\left(\mu - \frac{1}{2}\sigma^2\right)\Delta t + \sqrt{\Delta t}N(0,1)}$.

8 For example, GGY AXIS, Prophet, MoSeS.

9 The technical (algebraic) development of framework provides more rigor and insight into this and other concepts introduced in this article.

10 For example, AG43, C3-Phase 2.

11 The input structure could be two different tables, one for each product, for example. The key is the structure (no matter the number of constituent parts) is acted upon by the same throughput via unique entry points.

12 Obviously, the throughput would have different logic than would be the case if it were to support only a single product.

13 Similar to the input structure, nothing requires the output structure to be one "integral" unit (i.e., one table).

14 Attimu, Dodzi, and Bryon Robidoux. 2016. Abstractions and Working Effectively Alongside Artificial Intellects. *Predictive Analytics and Futurism* 14:18–23, *www.soa.org/sections/pred-analytics-futurism/pred-analytics-futurism-newsletter* (accessed Sept. 19, 2018).