REFLECTIONS ON MANAGEMENT
WITH TOM GALVIN

AUDIO TRANSCRIPT

Process Models and Other Oversimplifications (Knowledge Management, Part 9)

Season 7, Episode 9 – originally released 22 November 2022

Please note: This transcript has been edited for clarity.

> *Eight steps to doing organizational change. Ten steps to Improve Relationships. Six Steps to Becoming Your Authentic Self. We're fond of taking difficult tasks and breaking them down into predictable subtasks, stringing them together as lists so that it can be repeated and shared. It makes complicated things seem simpler, even when enacting them is anything but simple, and we have many ways to express such processes using boxes and arrows or numbered lists. But what about processes that cannot follow such a strict assembly line style model? It is not so much a problem that the steps are wrong, but that they tend to be incomplete because the context matters. So how do we address complicated processes without adding a lot of complications?*
>
> *My name is Tom Galvin and these are my Reflections on Management.*

So this is the final part of the **Describe, Explain, Decide and Act** continuum that I've been discussing in the majority of this series. And this is the one that would seem to be the simplest the action part, because that's one that we see a lot. What I gave is examples of X number of steps to doing something. That's something that we find in magazines and newspapers, in blog posts or whatever is the way we present complicated matters, trying to break them down into very, very simple, easy to remember digestible parts. We may even come up with mnemonics or acronyms that are supposed to help us with remembering things. But of course, that's only so helpful.

And yeah, you can probably take your typical process and express it in an assembly line fashion for those things where there is a defined input and a defined output and very little variation or opportunities for changing things as you go along. Okay, well, that's fine, but it doesn't work for every process. [I want to first cover] some options for looking at the simple and then bring in a model that I use for more complicated processes.

So I'm going to give three different simple models to show the contrast. Now, the first one is to start with the *numbered list*. The numbered list has an analogous form, which would be a horizontal block and arrow diagram. You know, in Western cultures, perhaps the blocks and arrows are arranged from left to right. Each of the steps means that the steps themselves are of somewhat equal footing. They're of equal value. You know, there may be some steps that are more critical than others, but the representation doesn't show that because the criticality of particular steps is not so important. What's

more important is the sequence that you start with a step one, you follow with a step two. Key is where are you in the process at any given time? What the block and arrow diagram represents is the movement of that information forward.

What's also important is this differs from the stream, which I talked about in the decision-making paradigm from the previous episode, because the process models are typically unidirectional. You know, you're thinking of from a beginning to an end, whereas the streams, each of the blocks, you're talking about independent context, independent decision spaces that are connected to each other. There's interdependence there and there's the potential of moving downstream and upstream, although moving upstream kind of goes against the flow here in a process model. Now what you're really thinking about is if you have a roadblock or you have some issue that prevents you from moving forward, you're not pushing back upstream so much as you're stepping out of the process and inserting yourself back into a previous state as almost like a do-over. And that would be like a feedback loop. You can think of that as a quality control action where a part on the assembly line obviously didn't get processed the way it should have. And so it gets pulled off the line, brought back to an earlier state in a similar fashion. The numbered list operates the same way because you're reading them from number one to number nine, and you really can't go backward so readily. You have to pull yourself out of the flow and reinsert back at an earlier step. If there's something that was missed or something that didn't work out.

Now, let's turn the block and arrow diagram sideways. Let's make it vertical. And this is one that in hierarchical organizations [e.g., a *chain of command*] will make a lot of sense, because when you're talking about taking a process model and saying that you're going to depicted vertically where the boxes are arranged in a column and the arrows are pointing downward, there's there is a sense of hierarchy that seems to be expressed with that. If the arrows are pointing upward, it seems to express vetting or permissions or collaboration informing. So the depiction, whether the arrows are pointed down or the arrows are pointed up, implies certain behaviors within the context of a particular organization that just make the flow natural, you know, the flow down of guidance, the flow down of direction, the flow down of expectations, requirements, etc., then followed by the upward return of recommendations, actions, whatever is a very, very easy thing to follow.

Now, sometimes we'll want to draw that diagram in a bidirectional fashion. And to capture both of the downward flow of guidance with the upward flow of the response. I personally prefer to draw them as separate columns because as a process model, the flow down and the flow up are discrete. The arrows, the flow of information should be depicted as discrete, even if it makes the blocks seem a bit redundant because it might be the same actors at the different parts of the process that are dealing with the information. The important thing is you're capturing the flow and if you show it as bidirectional, then it gets confusing because it seems like you only have to go down one level of the hierarchy and then the feedback comes back. It may not capture the full flow from start to finish.

Now more complicated [decision spaces], of course, can't be modeled with a linear structure, so we need something a bit more complicated. So, the third structure that I'll offer is the *flowchart*. It's another one that's very, very simple. You have a very common language that allows you to figure out

how to arrange a process so that it can handle branches or sequels of conditions-based decision making. So, the rectangles are a process where you're doing an activity, where you're doing something, When you reach a diamond, the diamond is a decision and coming out of the diamond are multiple arrows. If yes, go left, if no go down or something like that. The rhombus represents your inputs and outputs. When you see a rhombus, you're asking for information or you're providing information. And then, of course, the rounded rectangles. They're the terminals, the beginning at the start of the process and one at the end of the process, maybe multiple terminals, if there are multiple ways to derive at ultimately an answer and you start at the beginning and work your way through, those are the three forms that are familiar and that we use quite routinely.

Now, what happens if you really can't know the process up front? Using all of these models just does tend to assume that the process itself can be readily model, that the actions, the decisions or whatever are very discrete, that there is a recognizable flow from one thing to another and that the options are kind of limited. I mean, you can draw pretty complicated flowcharts, but then it gets very difficult to use it to communicate how things are actually working. If you have such a wide array of options, flowcharts tend to be fixed, which means that if the processes change significantly under certain conditions, then the flowchart doesn't help.

It's these sorts of cases that simple methods just are not sufficient. And so I have tended to go with another form of a model which I drew from computer science my early days in artificial intelligence. But really this is more pure computer science, and this is what we call the *finite state machine*. And the finite state machine allows you to consider the many ways that a process can manifest without having to try to diagram it in a simple fashion, because the process may be just simply too complicated for that to be possible. It also accounts for finality, which is that the process may very well produce a whole range of outputs or different ways of achieving the same output.

So the example that I'm going to give is that of a coin operated machine, a vending machine. So here's how the model works. We'll take a vending machine that is selling peanuts, candy, whatever and you want. You want to buy the peanuts and it costs $0.90. There's a coin slot that accepts nickels, dimes or quarters. So that's $0.05, $0.10 or $0.25. And there's a button to release the peanuts when enough money has been inserted and a button to cancel everything and return all of the coins if you change your mind. Now, there are many ways that coins can be combined to make $0.90 and the sequence doesn't matter. So at any time the machine is in a state that reflects the amount of money that has been put in so far. If not enough is put in, then the button will not do anything. The peanuts will not be released. If exactly $0.90 have been put in the button will release the peanuts and then that's it. Anything more than $0.90 and the machine will give back change based on whatever the overage is and give the peanuts.

The idea is that the finite state machines operate under a principle that there are acceptable output states and unacceptable output states. That is, some states basically require you to go on. The acceptable state is one that either provides the peanuts or the peanuts and change or returns all the coins. All the unacceptable states merely mean that something else has to be done because you haven't gotten all of the coins put in yet. The idea is. That you want to understand how the process

differentiates acceptable states from unacceptable states. Now, the difference from pure computer science, where you're trying to engineer the machine from the beginning here, it's kind of like we walk up to the machine as it's already in use and try to deduce from sort of reverse engineer what state we're in and what actions need to follow.

Without getting into all of the math behind it or the computer science, let me just talk you through the major elements of the finite state machine. And there are many, many variants of it. These are the basic ones. First is the alphabet, which is what are the inputs that the process will accept, which in this case is the coins and the choice to push one of the buttons. The states of the machine, the different ways that the machine could be configured at any time. In this particular case, we're really talking about mechanical levers inside the machine that is configured as each coin is inserted. And the placement of those levers not only determines the total value, but also how many of any particular coin have been inserted. So perhaps the state of having put in a quarter is different than having inserted five nickels. The initial state or the start point is where the levers are reflecting zero money inserted and peanuts are in the shelf ready to be dispensed. And when you reach the point at the end, when you've put in the money, gotten the peanuts or whatever it is, what the machine will reset to in order to achieve the to take care of the next customer is the transition function where you take a state and an action which is insertion, inserting a coin or pushing a button which defines what state the machine will transition to. Now, not only does this define the kind of movement through the process, but it also allows you to determine the costs of transacting with the machine or with the process. And then the set of final states or acceptable states which are the ones where you actually do achieve an output and which states. And that implies that the other states are ones where you have to continue to [provide more inputs].

You [ordinarily] want to map this out in its entirety in advance because that's what you're going to program the computer to deal with. But the way that I use it, this thought process is really more as an analogy and a forensic tool, looking at reverse engineering the process based on how the system behaves, trying to understand what it values as inputs, like what is the system's alphabet that drives the action, How does it define the various states that the system can be in, and what is the transition function and costs associated with that transition? If I see that, for example, that the system is stuck, then I'm looking at, well, how did we get here and why this was this particular path, this particular sequence of actions taken. Often it's because it appears to be either the most efficient path to success that turned out not to be, or there was a misunderstanding as far as what the actual state of the process would be after the action is taken.

Now, sometimes I may be able to reverse engineer all the way back to the initial state, or I know what the initial state is going to be. Maybe I'm seeing the output of the process being done and then the system resets and goes back to the initial state for the next action. So again, this is this is a process of reverse engineering, which means that I am examining only a portion of the system initially and being able to understand what the language is that the process is using and how it sees the movement. Since there would be many different paths to an acceptable output. Just like you can sequence the coins differently, then you're getting away from a deterministic model and thinking much more of a systems model.

The finite state machine metaphor is useful when we're talking about processes over information where there are in fact many different ways to achieve an end result because you can use information, the same information in different ways to achieve a result. An obvious example of this is the workaround. When you have something that's urgent and the ordinary process, the expected process will be too slow and whatever. There's ways of thinking about what would be considered an acceptable workaround, like what would achieve the desired outcome that would be different from what the process would ordinarily promise.

So then by reverse engineering that you can figure out things like is this workaround a potential new norm or is it so exceptional that it can't be replicated? What is it about the content of the action, The alphabet? The language in use that may make this case exceptional versus ordinary. How did the process work? How could it work better? And the language like this finite state machine analogy, allows you to think about how do you tweak the language, How do you change the way that the system understands the state, the steps or states that it is in? How do you change the language so that the range of inputs can change and you can modify the outputs? What about the range of acceptable states where perhaps a half solution is good enough, as opposed to what a regular process model would do, which is to produce a very firm output, and also how to reason about costs, what it costs mean. Because again, political standing, prestige, honor, a lot of intangibles can be wrapped up in cost that may allow one to think differently in terms of the efficiency of the process. This metaphor, I think, opens up the aperture for a whole different way of thinking about process modeling without introducing all of the complexity of complex adaptive systems. I think sometimes we confuse complexity with complication. I mean, complexity is about emergence. Complexity is about a system that's constantly changing, constantly in turmoil, lots of factors involved.

But when we're talking about process modeling, I think we're talking much more about complication, that we're talking on the level of bounding system activity to try to accomplish a very well-defined task of taking an input and producing an output without oversimplifying things all the way down to a linear model that is not really an accurate rendering of what's going on. So this is what I refer to as a process based meta narrative. It's the *process-based metanarrative* is the story of how we move from input to output in such a fashion. You know, that can include linear processes all the way to things that are complicated. It's not easy and perhaps not always recommended to try to draw the more complicated forms because then you end up with so many boxes and arrows, then you create confusion. A finite state machine representation just simply collects and describes the different parts, the alphabet, the states, what are considered acceptable states or whatever, perhaps more in a prose fashion or in a summary bullet type of a fashion may be sufficient. Generally, bullet list is what I use, supplemented with anecdotes and examples. And that's it. That's the last of the meta narratives. And so in the final episode, I'll wrap things up and talk a bit about what I took away from the whole experience of thinking through this knowledge engineering problem, making sense of that huge archive that I talked about in the first episode of the series. Hope to see you then.

And that's all for now. The views expressed are my own and do not necessary reflect the United States Army War College, the United States Army or the Department of Defense. Thank you for listening and have a great day.

All the Best!
Tom Galvin