Calibration of Complex System Dynamics Models
A Practitioner's Report

Rod Walker
Altus Consulting, Inc.
rwalker@altusinc.com

Wayne Wakeland
Systems Science Graduate Program
Portland State University
Portland, OR 97219
wakeland@pdx.edu

Overview -- The Challenge

This paper is not a typical academic paper that is solidly grounded in the literature. Instead, this paper reports practitioner's experiences in rebuilding and calibrating a very large system dynamics model. A prior version of this model had been in use for over 10 years in an ongoing executive training simulation. That model had never worked correctly in several key areas, requiring the outputs to be manually adjusted by very experienced facilitators during the course of the simulation. The present project rebuilt the system dynamics model, redesigned the parts that weren't working, and calibrated the resulting model to match the facilitator's experience in how it should work. The complexity of the underlying model dictated a relatively large amount of complexity in the replacement model. As desirable as it would for the model to be simple, such a model could not create the necessary degree of realism.

The case study model requires a large number of both inputs and outputs, and simulates a fictional company. It had to behave in ways that were consistent with the industry and the specific needs of the simulation model's learning objectives. There was no "real" company to reference in how it should behave. Another aspect to this case study was that there were a few parts of the old model that were clearly wrong, but the facilitators had become used to this incorrect performance. And in some cases, these differences significantly affected other parts of the model. This meant that there was no complete and consistent dataset for the reference behavior patterns that could be used for calibration purposes.

It was relatively easy to build a new model that corrected the errors in the old one. It was much, much harder to calibrate such a large model. In addition to the issue with so much unknown about how it should actually perform, there were 57 separate inputs and 296 outputs over the simulated 8-year period. All of these outputs needed to be in ranges that matched the expectations of the facilitators, and they needed to change in expected ways. For example, the rate of change in market share must be plausible (e.g., a firm in a mature market cannot gain 3 market share points in one year), and other metrics should change only over a long period of time.

In many of our models, there are typically only a few outputs that are of primary importance, and the others simply need to look “reasonable”. In the case study model, there was a need to get many different outputs into certain ranges in addition to getting the primary effects looking right. The size and complexity of the model, combined with the number of outputs, created a difficult problem to manage during calibration.

Validation/Verification/Calibration is a loop

Changes in one part of the model affect the other parts. In our experience, this has always been a consideration in calibrating system dynamics models. And this was a significant issue in the case study model because of its large size and its many relevant outputs – especially during the later stages of calibration. In a model this large, too many things affect too many other things. Changes to correct one issue frequently created the need for more adjustments elsewhere, creating a seemingly never-ending loop, as shown in Figure 1.

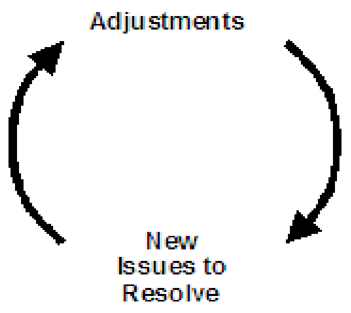


Figure 1. Model Adjustment Loop

The complexity of the case study model made the effect shown in Figure 1 much more pronounced. Some very long feedback loops affected most of the model. For example, quality affected sales, which affected production, which affected inventory, which affected importing, which affected quality (since the quantity and quality of the imported products affected the combined quality of products sold). Changes to parameters within parts of this loop could cause significant changes (and some unexpected ones) throughout the model. This can cause an exponential increase in the amount of work required, and in some cases could doom a modeling endeavor. An equally challenging situation can result even when overall model size and complexity is modest, but the model needs not only to be plausible in all respects but also needs to replicate empirical reference data for multiple model variables. A recent example of this type contained six state variables and a hundred auxiliary variables and parameters, and needed to simultaneously match reference data for five of the variables, leading to a nearly endless calibration loop.

As suggested in Figure 2, the amount of effort needed to calibrate SD models depends on both model size and complexity and on the approach utilized. Some approaches are easy to use for a simple model (Approach A), but can result in exponential increases in work as the complexity of the model goes up. For complex models, our strategies tend to look more like Approach B. This can result in more work for simple models, but limits the dramatic increases in work that can result when complexity increases.

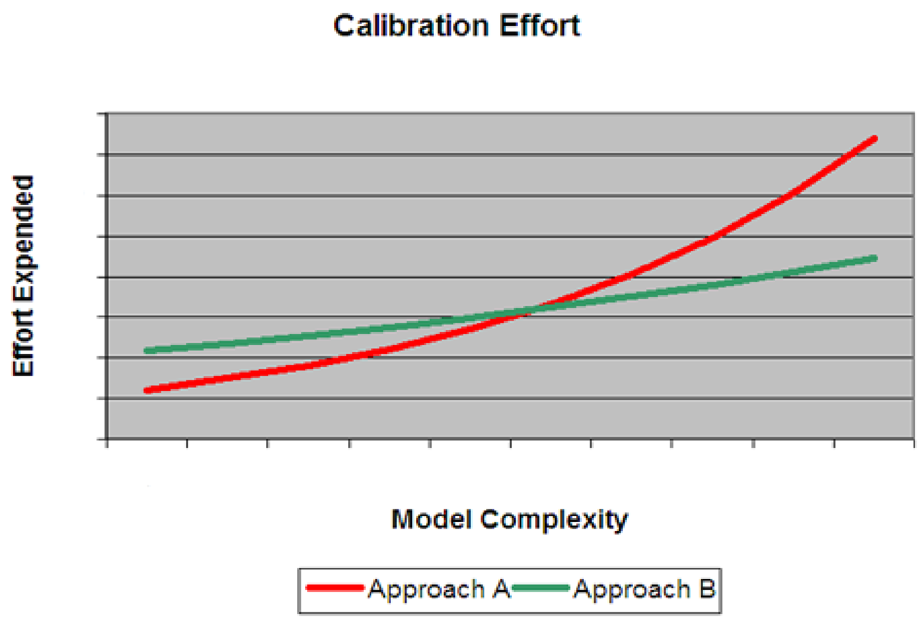


Figure 2. Calibration effort vs. model complexity and approach. Two different approaches are shown here (A and B). Both show an exponential increase as complexity increases, but with very different results for the most complex models.

Given all of these complications, it was critical to have a solid strategy. In fact, to get to a solution relatively quickly, the *strategy* seemed to be even more important than the specific techniques employed. The techniques which worked especially well are outlined in this paper.

Calibration Strategy

As noted earlier, calibration was not just an independent step, because changes to improve the calibration caused the need to change other parts of the model, frequently creating new calibration issues. So the strategy here was not just a "calibration strategy" but rather a "validation/verification/calibration" strategy. With this in mind, the following elements have proven useful:

1) Utilize traditional best practices as appropriate
2) Simplify the model as much as possible. Isolate interactions.
3) Redesign (instead of tweaking) when calibration becomes difficult
4) Document along the way to stay organized and minimize cycling
5) Know when to step away
6) Build or acquire automated tools to help in testing and analysis

These methods have been useful for all of our models, but they were especially helpful for the case study model. Within the above broad categories, a number of specific practices were especially useful, and are described below in some detail. Some of the practices are very easy and do not require much time, while others are much more time-consuming. But the time required increased linearly as opposed to the exponential increases in time that can result when attempting to use a simple calibration strategy with a complex model.

1. Traditional Best Practices

The literature (Sterman, Richmond, Barlas) provides a long list of useful techniques, and we find some of them to be particularly useful:

- Unit checking. This practice is very useful, especially as part of "code reads" and verification at a segment level. Along with unit checking, standardization of units has also been very important. For example, when using percentages, for external reporting reasons, we have standardized on making them always on a base of 0-100, and we put "pct" in the variable name. (For example "Bad Debt Pct" is the percentage of bad debt occurring, represented on a 0 to 100 scale). This practice makes it easy to go back through and verify that every "Pct" factor is divided by 100 before being used in the model. Of course, one could standardize instead on a 0-1 scale. The important thing is to have a standard for these types of variables that can always be relied upon when reviewing and debugging equation logic. This can save much confusion later in the project.

- Sensitivity testing. This tends to be especially useful at a model segment level to validate that smaller parts of the model are performing as expected, and that parameter value choices are robust (no "magic parameters").

- Transient testing. This also tends to be very effective, especially at a segment level. For example, if orders tripled in one month and then went back to the base rate the following month in the model, did the production and inventory logic function correctly? What if orders dropped to zero in one month and then went back to normal the following month? Simple tests like these can be very useful in identifying important errors in the model, and in building confidence in the model.

- Comparison Graphs (Richmond). The technique of using comparison graphs was very useful in making sure key parameters showed the right behavior and were in the right ranges. This required loading test cases and test results directly into the model, and creating graphs to make it easy to quickly compare the model behavior results with the expected or reference behavior. Entering these data and creating the necessary graphs can be a significant time investment when the model has many outputs, but the benefits far outweigh the costs. Another great use for the comparison graphs was to make quick visual checks for how the model was performing after changes were made. At several points in the project, parts of the model began oscillating, but the oscillations weren't obvious in the numerical results in the tables. The graphs quickly revealed the oscillations – an effect that might not have been obvious otherwise. These graphs can also be useful at a composite level as an overall check for oscillations. Frequently it has been useful to a create a single graph containing two or three key model outputs that are strongly affected by many different parts of the model, and to then check that graph frequently. For example, in the case study model, "Profit" is affected by nearly everything in the model in one way or another. So if there are significant oscillations going on somewhere in the model, it is likely that Profit will be oscillating as well. Keeping an eye on the graph of Profit can be useful as a quick overall check.

There are other best practices described in the literature, and we utilize many of them routinely without even thinking about them. We have noticed that the more complex the model, the more useful these best practices tend to be. Since these practices are well documented in the literature, the remainder of this paper focuses on specific elements of our calibration strategy that may have received somewhat less attention historically.

Simplification and Isolation of Interactions

Our experience agrees with the universal advice that models should be as simple as possible for the given purpose (Richmond, et al), and in this case that means including enough to make the training experience meaningful without over-complicating the model or the experience. For the case study model, extra complexity was inherited from the original model and could not be easily removed. Additionally, training simulation models tend to gain complexity as bugs are corrected and/or as calibration adjustments are made in different parts of the model.

As a general rule, our experience suggests that if a model is difficult to calibrate, the structure is probably the culpritThis characteristic seems to be especially pronounced for qualitative parts of our models, where if the structure is right, the calibration can go pretty quickly -- whereas calibration can be virtually impossible if the structure is designed without calibration in mind. And if calibration is eventually achieved despite a bad structure, it tends to be fragile.

As models become more complex, it becomes harder to spot structural errors. So it was very productive with the case study model to spend time continually simplifying the model. It also helped tremendously to temporarily break the long feedback loops during the debugging process. Specific techniques that were particularly useful included: A) submodels, B) temporary adjustment factors ("shims"), C) slowed transitions on inputs, D) simple cause/effect maps, E) careful validation at a submodel level, and F) independent validation and calibration of qualitative parts.

2.A. Submodels

In this model, one of the most valuable techniques was to take a troublesome section of the model and carefully redesign it until it was a simple as possible, then put it back into the larger model. The work with these submodels helped to clarify how certain parts of the overall model should behave, providing important foundation points when looking at the rest of the model. This was also a good way to "declutter" the larger model, because once the modeler determined which parts really mattered, it was possible to strip away unnecessary parts in the larger model. Submodels were also a great place to apply sensitivity testing as a tool to determine how robust the submodel was against or with respect to normally expected operating conditions in the model.

In deciding where to build submodels, we looked for parts of the model with clearly known expected outputs and behavior patterns which also provided main effects to the rest of the model. For example, there was a production planning piece of the case study model that had to regulate production starts in order to maintain certain levels of inventory despite changes in demand, changes in supply from other regions, and changes in exports to other regions. This part of the model quickly became very complicated through later changes, and it was very helpful to build a smaller, standalone version of this submodel and to test the key control logic before attending to the larger model. Boundary testing, transient and sensitivity testing was very useful at this submodel level in order to validate that it was providing correct responses and staying within the desired weeks of inventory. After completing the testing of the submodel, it was documented carefully and saved away as a separate model file for reference later. This exercise helped immensely in cleaning up the full model and in building confidence in this important part of the overall model. This technique was repeated in several other parts of the model when standard approaches didn't work. Each of these solidly tested submodels provided important points of stability in the overall model, making it possible to quickly rule out large parts of the model when investigating discrepancies[1].

The technique of building solid submodels was probably one of the most important keys to getting the case study model working properly. Given the size of the model, discrepancies in the outputs could frequently have been caused by a number of different parts of the model. Every tested submodel provided an important point of stability – something that could be ruled out (at least initially) when investigating these discrepancies. It saved a tremendous amount of time, and helped to minimize the risk of cycling[2].

2.B. Temporary Adjustment Factors ("Shims")

Builders frequently use shims to get all the separate pieces of a project aligned before finishing the construction, especially when parts interact, such as with doors and windows. Sometimes these shims remain in place forever, and sometimes they are removed as the construction is finished. In models, we frequently add "shims" in the way of temporary adjustment factors at the beginning of the calibration process. This helps to get the numbers in the right ranges, making it easier to see which parts of the model need more work. For example, if the market share is way off, many other numbers in the model will also be way off (production, revenue, costs, etc.). A useful way to see if those other pieces are correct is to force the market share into the right range, so a temporary adjustment factor was added to do that[3] (see Figure 3 for an example).

[1] A few times, the problem being worked on in the full model actually was created within a submodel that was thought to be solid. But in these cases, it was possible to jump back to the submodel file and documentation and quickly determine what had been missed.
[2] This process also provided an important mental benefit during the middle of the project. After finishing the testing of one of these submodels, it was very encouraging to feel good about the solidity of that piece, even if there were still numerous questions about the rest of the model.
[3] Another useful technique here was to temporarily use a SMTH function on inputs that were varying too much. See later discussion of this method.

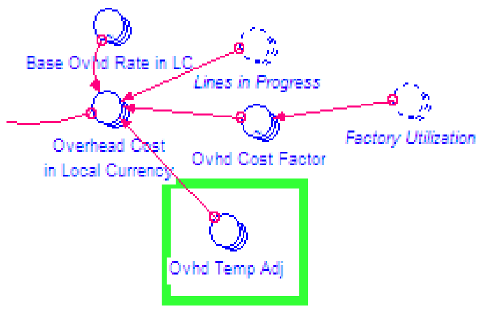


Figure 3. Temporary Adjustment Factors. These temporary elements were highlighted on the diagram with a bold rectangle to assure they would be removed or properly documented later

When a model is initially constructed, a particular approach to a certain part of the model might seem correct, but then during calibration the original concepts may begin to no longer make sense. Instead of reworking this part of the model (and possibly breaking something that was actually correct, thereby creating another endless calibration/rework loop), use of temporary adjustment factors was a huge help. For example, suppose that after considerable calibration effort, capacity remained at a figure that was consistently 5% below the correct value. As a temporary measure to allow work to proceed elsewhere we might add a temporary adjustment factor of 5% which is added to the calculated capacity. We would put a highly visible box around the factor in the diagram so it would not be missed later (see Figure 3). Once the rest of the calibration process was completed, these “magic parameters” could be safely removed. Without highlighting them, it is easy for these factors to become forgotten until they begin to create visible problems or confusion at some time later. (e.g., “Why is market share multiplied by 1.000234 here?”). At the end of the calibration process, these special adjustment factors were nearly all removed. If not, they were documented (and justified). Putting the highlight box around them helped to make sure they would not be overlooked later. These temporary adjustment factors have been especially helpful with parts of the model that interact extensively with other parts of the model.

2.C. Slowed Transitions Within Feedback Loops

In complex models, it is very easy for oscillations to appear in one part of the model due to changes made in different parts of the model. It can be difficult to identify the cause of these oscillations. One technique that helped was to temporarily slow down the rate of change around selected feedback loops. For example, within the case study model, exports were authorized based on production costs, quality levels, production capacity available, and shortage in the importing region. Exports started oscillating after revisions were made in the way product costs were computed. The oscillation was created by an intentional penalty on the producing region that would cause their costs to increase when they exported. The first time slot they exported, their costs would go up, causing them to no longer qualify. They would therefore stop exporting, and their costs would go down, allowing them to qualify again. So they were starting or stopping exports at every time step, leading to an artifactual oscillation (see Figure 4, below). A SMTH function on the cost comparison verified that this was the cause (see 5), and the model was redesigned to correct the issue.

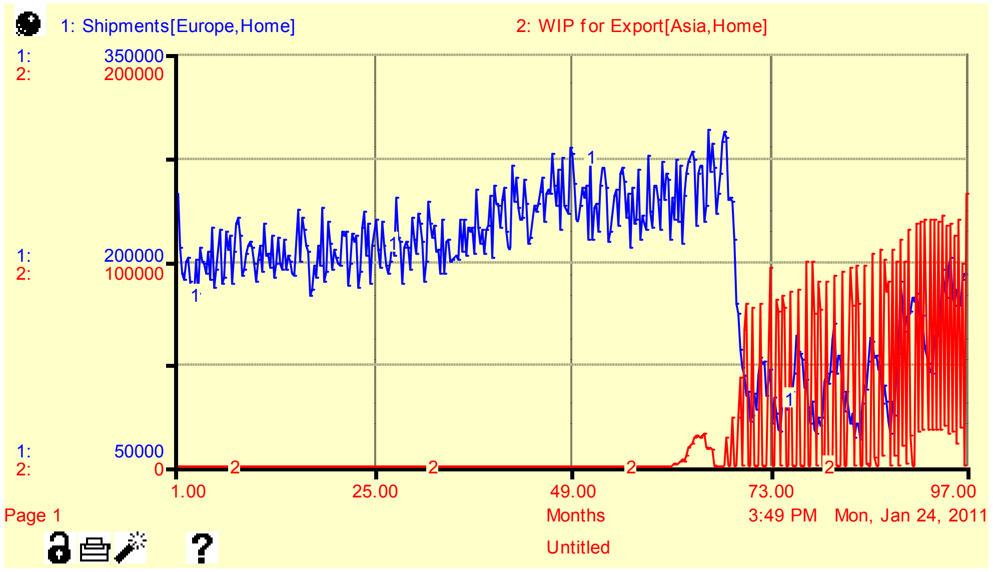


Figure 4. Oscillations. Note the oscillations starting prior to month 73 in WIP for Export. In this example, products are being built in Asia for sale in Europe (the blue line above). Variations in the blue line are caused by random fluctuations in market demand. The oscillations in exports caused shipments (in blue) to drop significantly.

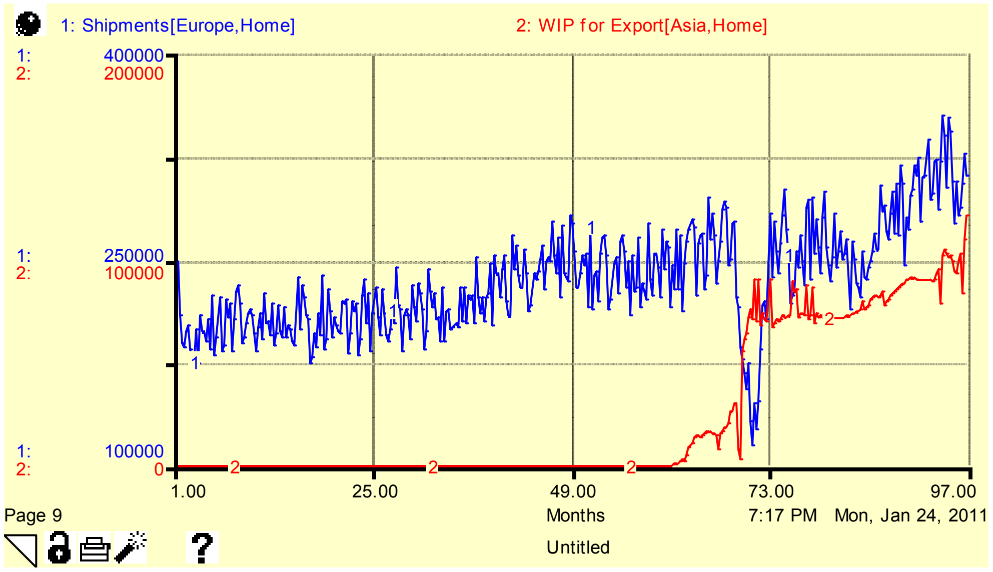


Figure 5. Oscillations Removed. After adding a SMTH function on the cost comparison, the oscillations were removed, and shipments (in blue) were back where they were

supposed to be. (The temporary drop in shipments at about time 70 was unrelated; caused by factory shutdowns in Europe prior to new factories being ready in Asia)

In this example, the addition of the SMTH was permanent, but prior to identifying the problem, temporary SMTH functions were tried in several locations to rule out other parts of the feedback loops as contributing factors. For example, quality could also have been a cause of the oscillation, because as imports increase, the effective quality level in the importing region changes based on the quantity and quality of the imports. As a quick check of this, a SMTH function was temporarily added to this quality input, and it was possible to quickly eliminate this factor as a possible contributor to the oscillations. Of course, there are other ways to arrive at the same conclusion, but this technique was both quick and effective.

Note also that these SMTH functions provided an easy way to temporarily get parts of the model into relatively steady state in order to be able to initiate further calibration and testing.

2.D. Simple Cause and Effect Maps to Isolate Issues

This one seems pretty obvious, but on a number of occasions while investigating model results it was helpful to remember to quickly sketch out a simple causal diagram around a troublesome output. For example, at one point the quality of shipped product was wrong. It could have been caused by too much or too little importing of product compared to local production and/or changes in the quality levels of the importing or exporting regions. A simple map laid all of this out, and then by examining the reported results for each of the causal factors, it was easy to identify the part of the model that had to be the problem. Then it was a simple matter to track down the reason.

This technique seems so simple that it could be easy to overlook and/or assume that it might not be necessary. But it was very helpful in restoring sanity at those times during long days when the model behavior had stopped making sense.

2.E. Validate, Verify, and Calibrate Initially at a Submodel Level

In a complex model, calibration of one part of the model can have ripple effects throughout the rest of the model, causing endless loops of adjustments in one area, then adjustments in another, then back to the original area. This is necessary to some extent with complex models, but techniques to reduce this looping really help. One such technique is to thoroughly calibrate segments as individual standalone submodels before trying to calibrate the entire ensemble. For example, in the case study model “Quality” was an important output on the reports, and it was also a critical feed into other areas of the model, showing its effect on production capacity, market share, etc. Calibration of something like revenue, then, could require endless iterations over capacity, market share, then quality, then back again. Instead, the components of quality were thoroughly calibrated first and reviewed with the client. The results of extreme conditions of inputs on quality were displayed through an automated Excel tool (see

Figure **6**), and adjustments of effects were made here until the client was satisfied with the responses. Then it was possible to treat quality as a "given" in the later calibration – so if one of the downstream numbers was wrong, it must have been because something else was wrong.

Similarly, market share was driven by a number of different factors (quality, advertising, service, price, etc.), and it was very helpful to look at extremes in these inputs and have the client decide if the responses were reasonable (see Figure 6). Calibrating these two areas (quality and market share) at a segment level significantly simplified the job of calibration of the overall model. Both of them needed to be adjusted slightly in the final stages of model testing, but the changes were small, and by then the other parts of the model were also reasonably well-calibrated. Other segments had fewer interactions with the rest of the model, but it was useful to get them to stable reference points as well. Then the job of calibrating the full model became more of a job of blending the effects of these different submodels (operating as black-boxes) without having to worry about changes inside the submodels.

2.F. Independent Validation And Calibration of Qualitative Parts

Qualitative parts of a system dynamics model can get tricky, and the case study model had two important qualitative components: first, a "Quality" measure which calculates outgoing product quality, and second, a number of factors affecting product market share – qualitative factors such as market awareness, sales effectiveness, service perception, etc. In the old model, both "Quality" and "Market Share" never worked correctly, and were manually adjusted by the facilitator during each workshop. Since these qualitative factors are quite subjective, it seemed important to get them locked down individually before proceeding to the rest of the model.

The first part to resolve was "Quality". Quality was expected to improve in more or less an S-curve as effort was expended, with a certain delay for actions to take effect. In addition to the S-curve shape, it should never go above a certain level (such as 85% -- different by product), and should take a varying amount of time to get there (different by product). Several other parameters/factors in the model impact quality, so, given the issues with quality in the old model, this part of the model was built and tested as a separate module, and the resulting behavior over time graphs were shared with the client (see Figure 6).

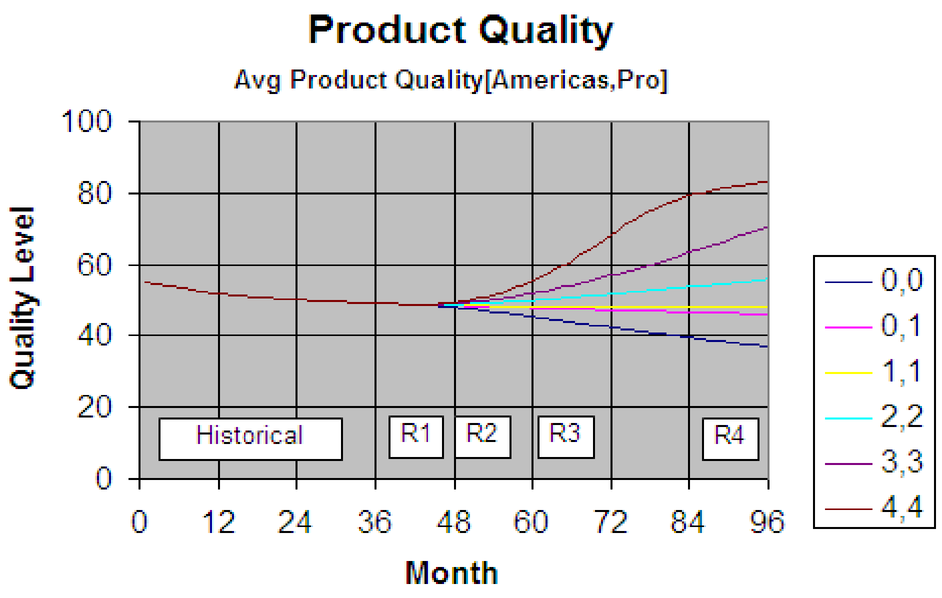


Figure 6. Qualitative Calibration chart showing a family of curves for different model input conditions. It was used to validate the behavior of this part of the model with the client.

Each trace in

Figure **6** represents the resulting quality for a specific set of input conditions. Similar graphs were created for each of the different products and regions, and the client reviewed each for expected behavior. For the remainder of the project, this part of the model could largely be ignored (trusted). It was important to be able to depend on this as being mostly correct, because changes here would ripple through large parts of the rest of the model.

The other qualitative/subjective area was in the market share logic and the six qualitative factors that drove it. These were similarly adjusted as a separate module (see Figure 7) before attempting to calibrate the entire model.

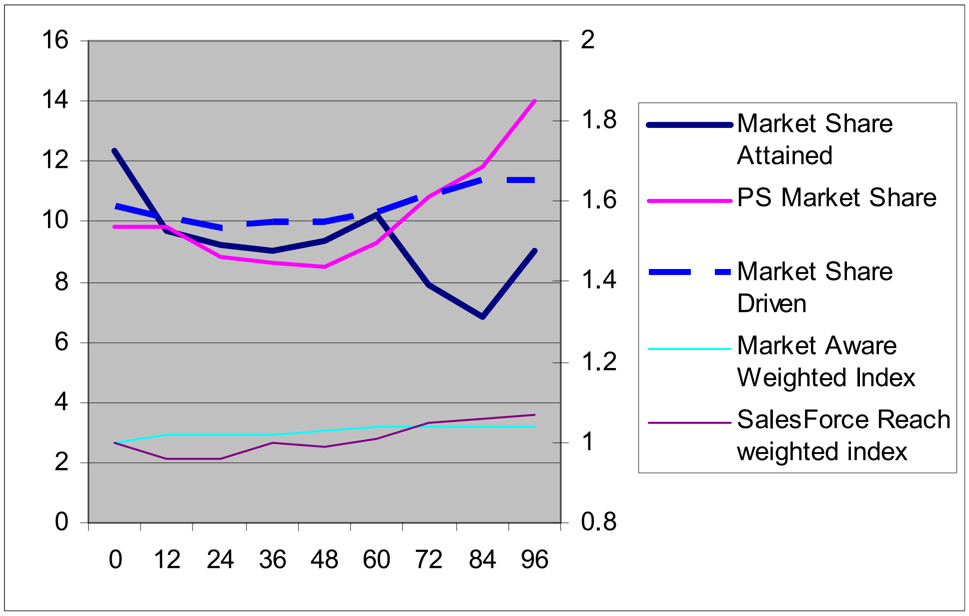


Figure 7. Market Share Example. For this product and region, the "driven" (demand) market share is shown in the blue dashed line, and the actual delivered market share is shown in the solid blue line. The difference between these two is caused by inadequate production capacity to meet all orders. The "PS Market Share" line in pink shows the market share produced by the earlier model. The factors at the bottom of the graph are two of the six components driving market share in the model. These factors were adjusted until the market share performance met client expectations for the given model inputs and state.

The ideas in this section were essentially parts of a "Divide and Conquer" strategy – aimed at breaking the big model into pieces that could be managed, and then solving as many problems as possible at the smaller level. Within that strategy, and within the rest of the model as well, continual redesigns and simplification were also important.

3. Redesign Along the Way

In our experience, when models are difficult to calibrate, it frequently means something is wrong with the design. It is very easy to design a model that is impossible to calibrate, and it also seems that models which are designed correctly calibrate quickly. Therefore, when calibration became difficult here, it frequently was productive to redesign part of the model. This was usually done through working with submodels as noted earlier, copying relevant parts to another model, simplifying it, validating it, then implementing the results back into the main model. Additionally, it was very helpful to just take the time to review related parts of the model as they were encountered in the debugging process, and carefully redesign them if appropriate. It seems like models invariably get cluttered up with unnecessary (and possibly undesirable) temporary fixes as a natural part of a lengthy project. Therefore, it seems helpful as an offset to this tendency to periodically look for opportunities to pull them out. In the case study project, following this practice resulted in the removal of a number of potential issues that would have been very difficult to track down otherwise.

4. Document Throughout The Process

“First Rule of Wing Walking: Don't let go of what you've got hold of, until you have hold of something else." (Anonymous)

As noted earlier, in a complex model, it is very easy for changes made late in the calibration process to affect other parts of the model that worked properly earlier in the process. As a result, the logic that is actually correct could get modified in an attempt to fix a problem introduced later. This cycling back and forth can be an issue with any model, but it was a particularly risky problem with the case study model, given its size and the number and the lengths of the feedback loops. In addition to the ideas mentioned elsewhere, careful documentation was a critical tool in minimizing counter-productive cycling through the model. Early cave explorers talked about tying a string, and letting it spool out behind them in case they needed it to find their way back out. Early wing-walkers learned to always keep one hand (or at least one foot) on the wing. Good documentation in a complex model can provide the same type of safety net. Specific techniques for improving documentation include: A) revision management, B) recordkeeping, C) code reads, and D) maintain a questions list.

4.A. Revision Management

We have always found it helpful to maintain a careful process of model revisions management. Our system is to save away the first version of the model as something like “Model rev A1.itm”. When a number of changes have been accumulated, we would save the updated version away as “Model rev A2.itm”. This process continues throughout the project. When significant changes occur, we might go to rev B1, C1, etc. As a further example, if we are working on version A23 and made a big change to how demand is generated, that would be a signal to move to A24. If we decide later to change the demand generation part back, along with a new set of different changes, the revision would be incremented to A25. That would make it possible to later recover the method used in A24, if it was decided that it was a better solution.

The important part of this process is to maintain an appropriate degree of discipline, and to jot down notes to indicate what changed in each particular revision. This practice has been a huge help with large models that might be under development for weeks or months. This practice also has the obvious potential to serve as a sort of macro “undo” function, making it possible to revert to an earlier revision or to be able to recover pieces from an earlier version. But it was also very helpful in just understanding why something that used to work no longer worked. (For example, “Inventory management worked fine in A22, and it’s no longer working in A29, even though I didn’t make any changes in that area. Why?”) It is a very quick process to pull up each revision of the model and compare the relevant pieces. If necessary, it is also possible to save the equations in a text file and use text file comparison to identify all of the changes. (see Figure 7). Saving many model revisions is a very easy thing to do, and it can produce huge benefits.

4.B. Recordkeeping

Keeping a change log is an obvious part of documentation, and yet it is easy to overlook. In the case study project, recordkeeping involved keeping meticulous notes about what changed and why. This made it possible to answer the question later of "what was I thinking when I did this.” It was very useful to index this by model revisions as noted above. It can include the changes in a given model revision, why they were made, relevant assumptions, etc. Without this information later, it can be hard to decide whether what I’m looking at is just a bug, or was done that way for a particular reason. That information can save a lot of time trying to figure it out – or worse, creating a new issue by throwing out something important. The hard part about recordkeeping has always been remembering to take the time to make a few quick notes before moving to the next thing. The changes and the rationale all seem so obvious at the time that writing it down seems unnecessary. It isn’t. It can also be helpful to include in the change log screenshots of model runs, references to or extracts from information sources, quotes from telephone conversations, etc.

4.C. Code Reads

Related to some of the ideas mentioned earlier, “code reads" were very helpful in finding and removing errors without having to trace back from the output. This also was a good way to clean up parts of the model that had become confused by various changes over a long period of time. The process of read, analyze, rewrite, test helped to tighten up the model and make it more sustainable.

Since our models are frequently part of ongoing training programs, they tend to be used for years. One business simulation has been in continuous use for over 5 years in an MBA program, with several thousand students having gone through it to-date. With that many students trying to find a way to “beat” the model, it is imperative for the model to be robust and well-tested, and these code reads (and rewrites) really help to achieve that goal. It is also helpful when questions and/or issues arise years later because clean code makes it more likely that answers can be quickly found. The documentation described earlier can also be very helpful in these situations, but having a clean model seems to be even more important than having great documentation.

Note that these “code reads” are different than just reading back through the code while debugging. The biggest difference is in the mindset, at least for us. It is too easy to simply see what you expect to see. In the debug process, you’re looking for a bug, and the more correct code you see, the more likely it is that you subconsciously begin to assume that everything you see is correct. And therefore, that is exactly what we see. (In these situations, an outsider is frequently able to spot the bug more quickly). A “code read” is an examination of the code without knowing there is a bug there – seeing if it all makes sense and is as clean as it could be. Effectively, the mental assumption during code reads is that “everything is wrong unless proven otherwise”, as opposed to “everything is OK unless I can see how it could have caused this bug.”. It seems to us to be a very useful mental shift. By doing a code read looking for weak links in the chain of logic, it has been possible to spot something like a subtle units-conversion problem which might have been difficult to spot during debugging. This type of error can often be found in a careful code read, avoiding a potentially lengthy debug process when it shows up later in testing and calibration.

As changes are made during the development and debugging process, it is easy for the model to become cluttered with quick fixes that may or may not have been appropriate. And, as the overall model evolves and matures, some of these fixes may have become no longer necessary, and some may actually get in the way of progress. The code read process helps to identify these fixes so that they can be removed. Overall, the code read process can help tremendously to get a model slimmed back down to its essential parts – especially late in the project. This helps to make the model more robust, more explainable, and more maintainable (and less embarrassing when someone else reviews your work)!

4.D. Maintain a Questions List

In projects like the case study project, the modeler will invariably have numerous questions from day one. A useful practice is to note these questions in the project notebook as they arise, with an open check-box beside them. Some of them will be answered quickly, and can be easily checked off. Many remain for a while, because the answers are not obvious, and other work can be performed in the meantime. As questions are resolved, they are checked off, and the answers are noted in the project notebook. Some of these solutions will need to be revisited later, so the documented explanation can be invaluable. Or, during debugging, the question might come up again. In both cases, it can be very helpful to have a record of the questions and how they were answered[4].

[4] These questions (and solutions) can also be very helpful when creating the final user documentation for the project. The list is a great way to remember important assumptions that were made in the project and other questions that a user might have about the behavior of the model. If we have a question when building the model, others are likely to have similar questions at some point.

Although the case study model was not created from scratch, a number of questions were noted as the new model was built. These were helpful in identifying places to start in the debugging process, and in reminding us of places that needed more thought after the entire model had been implemented.

Further, writing down issues can be intrinsically helpful, because in trying to explain the problem in words it can help to reveal errors in thinking or bad assumptions.

5. Know When to Step Away

We all know that there are times when the best thing to do is to just step away from a complex task for a while. But it's hard to recognize when to do this, and even if it is recognized, there is a mental battle between the need to take a break from the model versus the concern to keep working while the model is all "in our head." In the case study project, when the modeler finally decided that it was time to step away, he wrote down everything he had observed about the current problem and any thoughts about what to try next. Then he took the break. After taking the break, he nearly always had an immediate insight about where the problem might be. And if not, at least there was a good starting point for getting back on track. Stepping away was helpful, and the documentation was useful then and later.

6. Build/Acquire Automated Tools

There are times when system dynamics model development feels a lot like electrical engineering. In that field, it is frequently necessary to design and build devices that are only used to help check out the design of a particular product. It can seem expensive to spend the time and money to construct this test hardware, but it is frequently essential. In system dynamics models we have seen the same need from time to time. Fortunately, the SD development platforms provide helpful tools for this task, but it might be necessary to make a significant time investment. Although the cost might be high, the payoff can be huge. Specific techniques include: A) automated testing tools, B) automated analysis tools, and C) code comparison utility.

6.A. Automated Testing Tools

The SD development platform usually provides a method for doing sensitivity testing, and sensitivity tests are essential for validating the stability of submodels, as noted earlier. Sensitivity tests are also helpful for studying the results of combinations of the many different inputs to the overall model. However, preparation of the inputs for the sensitivity testing could easily become time-consuming and error-prone. In this project, several times we built special Excel-based tools that would create the inputs for these sensitivity tests and assist in analysis of the outputs. For example, this method was used to create the quality profiles shown earlier (e.g.,

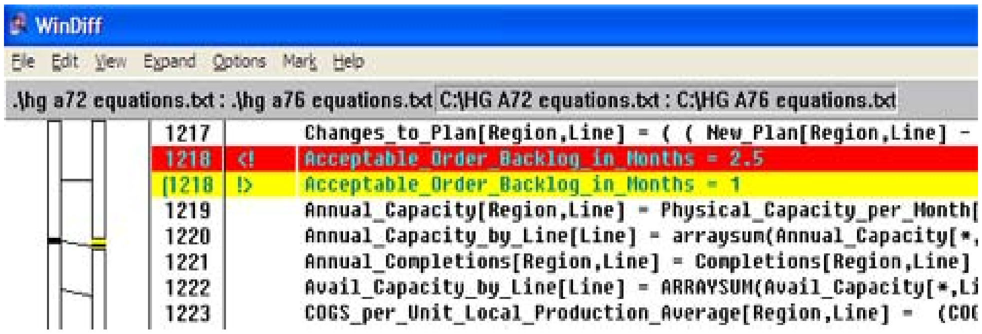
Figure **6**).

6.B. Automated Analysis Tools

In addition to creating automated testing tools, we also created tools that were just used in helping to analyze the results. As noted earlier, the case study model had 296 outputs that were important, and it was easy to miss undesirable changes in some of those outputs while other parts were being adjusted. An Excel-based tool was constructed that would compare the results from a number of model revisions and show the differences. This tool looked at each time slot of all 296 outputs for each model. The tool was a bit tedious to construct, and for that reason, it was not built until late in the project (out of necessity at that point). But it was well worth the effort, and in hindsight it should have been built much earlier.

6.C. Code Comparison Utility

A tool borrowed from the software development realm can be useful for identifying differences between various model revisions, as noted earlier. One such tool is WINDIFF.EXE, provided by Microsoft[5]. This tool quickly shows every line that is different between the two files, making it easy to quickly see how the two versions differ.



```
.\hg a72 equations.txt : .\hg a76 equations.txt C:\HG A72 equations.txt : C:\HG A76 equations.txt
 1217         Changes_to_Plan[Region,Line] = ( ( New_Plan[Region,Line] -
 1218   <!    Acceptable_Order_Backlog_in_Months = 2.5
(1218   !>    Acceptable_Order_Backlog_in_Months = 1
 1219         Annual_Capacity[Region,Line] = Physical_Capacity_per_Month[
 1220         Annual_Capacity_by_Line[Line] = arraysum(Annual_Capacity[*,
 1221         Annual_Completions[Region,Line] = Completions[Region,Line]
 1222         Avail_Capacity_by_Line[Line] = ARRAYSUM(Avail_Capacity[*,Li
 1223         COGS_per_Unit_Local_Production_Average[Region,Line] =  (COC
```

Figure 7. Text File Comparison Tool. The screenshot above is from WINDIFF, showing part of a comparison between two model revisions (A72 and A76) – one in red and the other in yellow. All lines that are not colored are identical between the two files. The scrollbars to the left show where in the file differences appear.

[5] See the Wikipedia entry for WINDIFF or the Microsoft website for more information.

This tool has been especially useful in tracking down why part of a model suddenly no longer works, even though it appeared to work earlier. It is a simple matter to go back and get text versions of the equations from each of the two model revisions, and then the comparison tool will quickly show everything that is different between the two model revisions. This simple technique has undoubtedly saved hours of confusing analysis. Since it is so easy to use, it has also made it possible to investigate and identify the causes of special conditions observed within the model that aren't clearly "bugs", but just aren't working the way the way it was expected under different inputs.

The automated tools described here are not always very important in simple modeling projects, but in complex ones, we believe they can make a huge difference in the time required and the quality of the resulting model.

6. Conclusions

Every system dynamics project offers the opportunity for learning, and the case study project offered many such opportunities. The project served to validate a number of our normal practices, and it also inspired the creation of new ones. A few general themes seemed especially important.

First, many of the techniques presented were clearly valuable in hindsight, but there was a strong temptation to not do them originally because they seemed like unnecessary work. Some techniques might not have been necessary on a small project, but all of the techniques were very useful (in fact essential) on the case study project, and it would have been helpful to have made the investment earlier.

Second, staying organized on a project like this was a major challenge, and many of the tools discussed earlier were very helpful. Thorough submodels provided important points of stability, making it easy to decide "the problem is elsewhere," which helped to preventing throwing out solid work by accident. Careful documentation and revision management provided a stable reference point on what was done, why, and what questions remained.

Third, having a clear strategy was critical. Simple models, or models with only one or two key issues, can be approached without worrying so much about a strategy. But complex models that might have many critical issues can create endless cycling unless a good strategy is carefully followed.

Finally, continual redesigns were critical in the process, especially given the way the case study model will be used. It was a lot like the military idea of "divide and conquer," where a large opposing force is conquered by concentrating overwhelming forces on smaller units one at a time. In this project, it was more like "divide and solidify" – put all of the focus on a small part of the model, and don't leave until that small part is solid – then move to the rest. Initial versions of our models are pretty clean, and invariably get cluttered as changes are made to correct various issues or to solve specific client needs. By taking advantage of opportunities to reexamine or redesign pieces of the model along the way, the final model is once again clean, providing a stable and understandable result. And the process also avoids having to track down innumerable issues later. Additionally, this practice is also extremely beneficial months or years later when there are questions about the model. Many of these training models are used for years, and they generate an ongoing stream of questions from instructors and students along the way. Because the models are clean and well-documented questions can often be answered in 15 or 20 minutes, even years later – and half of that time is spent reconstructing the situation in the model that motivated the questions.

In the case study project, and in our experience in general, validation/verification/calibration of system dynamics models is a loop, as illustrated in Figure 1. Changes made as a result of issues in one of these three activities often affects the other activities, creating the potential for endless cycling. We break these cycles by using a divide and conquer approach, getting each piece solid, simplified and well-tested before moving to the next. And we assume that if calibration is difficult, the cause is most likely within the model structure. This overall strategy, along with the specific techniques described, was very helpful in getting the case study project to a completed state 10 years after calibration work on the earlier model was largely abandoned.

References

Richmond, Barry. 2001. *An Introduction to Systems Thinking*. High Performance Systems, Inc.

Sterman, John. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World.* Irwin McGraw-Hill.

Barlas Y. (1996). Formal Aspects of Model Validity and Validation in System Dynamics, System Dynamics Review, 12, pp. 183-210