Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

MODEL VERIFICATION AND VALIDATION

John S. Carson, II

Brooks-PRI Automation
1355 Terrell Mill Road
Building 1482, Suite 200
Marietta, GA 30067, U.S.A.

ABSTRACT

In this paper we outline practical techniques and guidelines for verifying and validating simulation models. The goal of verification and validation is a model that is accurate when used to predict the performance of the real-world system that it represents, or to predict the difference in performance between two scenarios or two model configurations. The process of verifying and validating a model should also lead to improving a model's credibility with decision makers. We provide examples of a number of typical situations where model developers may make inappropriate or inaccurate assumptions, and offer guidelines and techniques for carrying out verification and validation.

1 WHAT IS VERIFICATION AND VALIDATION?

Models are used to predict or compare the future performance of a new system, a modified system, or an existing system under new conditions. When models are used for comparison purposes, the comparison is usually made to a baseline model representing an existing system, to someone's conception of how a new or modified system will work (i.e., to a baseline design), or to current real-world system performance. In any of these cases we want to know that the model has sufficient accuracy. Sufficient accuracy means that the model can be used as a substitute for the real system for the purposes of experimentation and analysis (assuming that it were possible to experiment with the actual system).

After the first phases of a modeling project – the kickoff meeting, the functional specifications document, and initial model development – the model developer enters into a series of activities called debugging, verification, model review and validation. In fact, writing and seeking approval of the functional specifications document lays the foundation for verification and validation; this is where all modeling assumptions and data requirements are laid out for the project team to review, modify if necessary and approve.

In practice, a model developer intermixes debugging, verification and validation tasks and exercises with model development in a complex and iterative process. We separate them conceptually for explanatory purposes and in the hope that in practice verification and especially validation do not receive less time and effort than their due.

1.1 Definitions

Debugging occurs when a model developer has a known "bug" in the model and uses various techniques to determine the cause and fix it.

Verification occurs when the model developer exercises an apparently correct model for the specific purpose of finding and fixing modeling errors. It refers to the processes and techniques that the model developer uses to assure that his or her model is correct and matches any agreed-upon specifications and assumptions.

Validation occurs when the model developer and people knowledgeable of the real system or new/modified design jointly work to review and evaluate how a model works. It refers to the processes and techniques that the model developer, model customer and decision makers jointly use to assure that the model represents the real system (or proposed real system) to a sufficient level of accuracy.

The verification and validation phases often detect bugs that require further debugging, or incorrect assumptions that require significant model modifications and then further model re-verification and re-validation.

It should also be noted that no model is ever 100% verified or validated. Validation is not an absolute. Any model is a representation of a system, and the model's behavior is at best an approximation to the system's behavior. When we (loosely) say that a model has been verified or validated, we mean that we have explicitly carried out a series of tasks to verify and validate our model to the degree necessary for our purposes. Such V&V is always a matter of judgment to a large extent.

Model *credibility* refers to the decision maker's confidence in the model. One goal of the V&V process is to gain this credibility.

1.2 Responsibility, Attitude, Aptitude

In modeling projects for the industrial and service sectors, all except the largest projects typically are carried out by one, or sometimes two, model developers. Although there are others on the modeling team – engineers, process owners, operators and supervisors, decision makers and managers, vendors, designers, and systems integrators – the model developer has the primary responsibility to assure that the model is accurate, and to correct deficiencies as they are found during the verification and validation process.

On occasion I have found an attitude among some model developers that an in-depth investigation and review of *their* model is viewed as a review of their job performance, or worse, an attack upon them. In fact, a third party review is often essential to have reasonable assurance of model accuracy. In professional simulation consulting groups, it should be common practice for a more senior, experienced model developer to review in detail models developed by the newer model developers. In this and other ways, the model developer's attitude plays a critical role in the quality of the model verification and validation exercise.

It has also been my experience that some model developers, especially newer and more inexperienced ones, find the V&V process difficult because it requires a change in mind-set and attitude from that required when developing a model. As the well-known statistician George Box has said: "All models are wrong. Some are useful." So while verifying models, it is good advice to keep the first part of Box's quotation in mind, in order to maximize the likelihood of the second part becoming true – and to keep a sense of humor and perspective about one's own creative (model building) activities and the all-too-human tendency to overlook errors in one's own work. For large and complex models, the model building process becomes challenging and intense, engendering a sense of "protective" ownership in the model – good in itself but sometimes counteracting the need for thorough model testing. "Engineers ... are not superhuman. They make mistakes in their assumptions, in their calculations, in their conclusions. That they make mistakes is forgivable; that they catch them is imperative. Thus it is the essence of modern engineering not only to be able to check one's own work but also to have one's work checked and to be able to check the work of others." (Petroski, 1992)

Indeed, it is useful if a model developer approaches debugging, model checking and model verification with the assumption that their model is wrong, even after all obvious bugs are fixed – and that their job is similar to a medieval inquisitor. Simply because there's no apparent evidence that you have done anything wrong (made a modeling error) does not mean that you are innocent (the model is valid). It's guilty until proven innocent! It's your job to find any weak points, to get to the bottom of any problems, to "stress test" the model, to break it – and to find the problems and fix them.

At all costs avoid the "student syndrome": If it's late at night, and it runs to completion, and it's due the next morning, print out the results and turn them in. (My apologies to all the excellent students I've had over the years – this is not you!). This could also be called the "one run looks OK, so the model is correct" syndrome.

Other perspectives on verification and validation can be found in all past Proceedings of the Winter Simulation Conference in the Introductory Tutorials track and often in other tracks. We mention the most recent articles: Law and McComas (2001), Sargent (2001), and Schmeiser (2001). Simulation textbooks that discuss verification and validation include Banks et al (2001) and Law and Kelton (2000).

2 CATEGORIES OF MODELING ERRORS

Modeling errors may be grouped into the following categories:

- Project Management Errors
- Data and Data Model Errors
- Logic Model Errors
- Experimentation Errors

We discuss each of these in the following sections.

2.1 Project Management Errors

These errors revolve around project management and related communication issues. A simulation model involves a team: the model developer(s), the customer or end-user, and often-times equipment vendors, other consultants or engineering and design firms. The customer side may include engineers, maintenance personnel and other support staff, as well as line staff (operators and supervisors). The customer side should always include managers and decision makers.

It is critical for project success to have all the key personnel involved from the beginning of the project.

Example 1: At the kickoff meeting, only the engineer designated as the key person was present. At a subsequent review meeting, other people were present. On reviewing key model assumptions, some of the other personnel disagreed with assumptions. As the assumptions involved the areas in which they worked, their opinion was accepted by the group. Unfortunately this led to some project delay as the modified assumptions had to be incorporated into the model.

Example 2: A manager who had not been present at earlier meetings came to a model review meeting. He disagreed with some key assumptions that had previously been made, and subsequent discussion with the group concluded that he was correct on some points. As a result, the model developer (whether internal or an outside consultant) had to spend unexpected time making significant changes to the model, resulting in project delay and/or additional project costs.

The moral is simple: Get all people involved from the beginning. Get all to agree to and "sign-off" on the model's assumptions. Get all to agree to the objectives of the study, and to agree to at least an initial list of questions to be addressed. Make it clear which questions or areas cannot be addressed because of the model's scope or level of detail. In the kickoff meeting and functional specification document, emphasize the importance of accurate data for resulting model accuracy and validity.

2.2 Data Errors and Data Modeling Errors

Data errors are errors in the input data itself, in the form of the data, or the completeness of the data. Data modeling errors refer to errors in how the data is used in the model, a typical one being to assume an inappropriate statistical distribution for random data.

Some typical sources of data errors include:

- Only summary data is available, when what is needed are individual values. For example, a machine's nominal processing rate is all that is readily available, but in reality processing times vary for numerous reasons.
- Data may be grouped but individual values are again not available. For example, total number of downtimes and repairs per shift for all machines is collected but not actual running times and repair times for each machine.
- Records may show when a machine is down, but may not give the cause or reason. For example, a machine that is not needed for 4 days may not be repaired for 3 days, but this does not mean that 3+ days is a typical time-to-repair.
- For equipment that fails, one value called machine efficiency is available when what is needed is some measure of individual times-to-failure and time-to-repair. Knowing only that a machine is 94% efficient (or available or productive) does not tell you whether times-to-failure and times-to-repair are long or short. Longer TTF and TTR, while resulting in the same individual machine efficiency as shorter ones, may result in much different system performance.
- The customer says that all kinds of data are available. It turns out that the available data corresponds to simulation outputs – items such as throughput, response time to a special order, overall system response time – and not to simulation input data.

Some typical data modeling errors include:

- Using a mean when actual values vary randomly.
- Using a mean when actual values vary by some attribute (such as processing times that vary by part type).
- Using a statistical distribution simply because a time varies when variability is due to known causes
  - Example: processing time at a machine or station varies greatly for each part type. Parts arrive in batches all of the same type. For a given type, processing time is virtually constant.
- Modeling machine failures inappropriately:
  - In some cases, number of cycles between failure, or busy (running) time to failure, is more appropriate than clock-time-to-failure.
  - When detailed data is not available but the cause of failure (or product jam or other stoppage) is due to tool wear-out or any other fairly regular event, using an exponential distribution may not be appropriate.
  - When detailed data is not available, assuming a normal or uniform or triangular distribution (or any other symmetric distribution), simply because of familiarity, for time-to-failure is usually totally inappropriate and invalid.
  - Adjusting the processing rate to account for the failure rate (and ignoring the up/down machine states associated with actual failures) is almost always inappropriate and results in an invalid model.
- Assuming statistical independence when it is not appropriate
  - Example: For an order fulfillment center that had both very large orders (many line items, large quantities) and small orders (often one item), orders were modeled by two random variables: number of line items and quantity for each line item. A statistical distribution was chosen for each, but any dependence was ignored.

For this last example – order data – it is usually the case that using actual data when it is available is better than some statistical summary of the data. When needed for experimental purposes, larger order files for a given period (such as one day) can be obtained by combining orders from multiple days or by random sampling from a

given file of orders. With such data there is often a complex correlation between number of line items on an order and quantities. Simply put, it is likely that large customers order large quantities of many different items, and small customers just the opposite; and it may be most likely that an order for one item has a quantity of one. Therefore, sampling independently for number of line items and the quantity of each line item typically leads to an order profile much different from the actual one. (The statistical approach may be adequate for order data depending on it nature and provided the data is stratified. That is, characterize large, medium and small orders separately; within each grouping of similar orders, the two variables mentioned may provide an adequate representation of the given order profile.)

2.3 Logic Modeling Errors

Logic modeling errors refer to errors in the model's specification or implementation in the simulation language.

Some errors are language-dependent or common to languages with similar concepts and implementations. For example, in Schriber et al (2001), Jim Henriksen says: "Faulty indexing is the number one source of computation errors in GPSS models." This comment potentially applies to any simulation language that uses variable arrays and arrayed entities to facilitate modeling of parallel activities and processes. While this type of error is not likely to occur in smaller models, it is not uncommon in larger, more complex models. It points to the need for a model developer to aspire to become an expert in his or her chosen simulation software.

Many logic modeling errors fall into the category of poor or wrong assumptions, often brought on through faulty project management.

Example 3: In a model review meeting where the end customer was present for the first time, the animation showed an order picker with one tote moving down a pick aisle. The first comment from the Director of Engineering was "It doesn't work that way." It quickly became clear that a picker typically pushes along 3 to 4 totes, picking into all simultaneously. The problem lay with the engineering firm that outsourced the simulation model development; they had "made assumptions" without observing closely enough the workings of the actual system and conveyed faulty assumptions to the model developers (who were not able to have any contact with or observation of the end customer's distribution center).

A broader and often overlooked source of potential errors is due to a poor model design or a poor approach to one aspect of the model. Many modern simulation languages allow a great deal of flexibility in how any given situation may be modeled. This is good in that it increases the power and scope of the tool, allowing for accurate modeling of unusual and unique situations, but it provides a challenge to newer, more inexperienced model developers. It is also true that the approach taught for the simple models used in many simulation classes (including vendor's introductory classes!) may not be easily "scalable" to large real-life models.

For example, in most process interaction simulation languages, the natural approach to modeling a limited resource is that of an "active object, passive server"; this leads naturally to a "push" philosophy for managing jobs or tasks flowing through the system. The active object may be called an entity, transaction, or load (depending on the simulation product being used); the passive server may be called a resource, facility or storage. In some modeling situations, an "active server" approach may be more appropriate in the sense that it leads to code or logic that is easier to develop, debug, verify and modify.

Example 4: In a job shop model using a "pull" philosophy, a new model developer used the natural "active object, passive server" approach that she had learned in an introductory class. The active object was a part being produced; the passive server was a processing machine or workstation. This made it difficult to implement the desired "pull" control strategy at workstations as well as the integration of work-in-process buffers into the control strategy. In fact, an "active object, passive server" approach leads naturally to a "push" philosophy. With the suggestions of a more experienced modeler, the new modeler changed her approach and found it much easier and more straightforward to implement the desired control strategy. This ease and straightforwardness carried over to the debugging and verification phases.

Example 5: In a model of an automated guided vehicle system for a semiconductor plant, a model developer took an initial approach that resulted in vehicle control logic being dispersed throughout the model. This initial approach made it difficult to communicate the logic to another team member as well as to verify its correctness. A person more experienced in the simulation product being used pointed out that there were at least two other approaches to modeling the control logic that put all the vehicle control logic and decision making into one compact procedure. This alternate approach was based on an "active server" where the server represented the control software system which was notified of relevant vehicle events and responded appropriately with vehicle commands. The new approach resulted in compact vehicle code that was easier to develop and verify. In fact, the new approach made it possible for the controls software team to review the simulation code representing their controls. (Even though the controls programmers had not been trained in the simulation software, their general programming experience made it easy for them to read the simulation code for vehicle control and verify its correctness – because it had been written in the compact manner described above, and secondly because some limited programmer's documentation such as variable definitions was provided.)

2.4 Experimentation Errors

Some common errors during the experimentation and analysis phase of a project include:

- Not understanding the need for statistical analysis
- Too few runs
- Not understanding statistical sampling error
- Failure to do an adequate warmup determination for steady-state analyses
- Misinterpretation of confidence intervals

These and similar errors are discussed in detail in Schmeiser (2001). Although these errors are typically not included in a discussion of verification and validation, they may lead to a model not being useful and indeed to "garbage in – gospel out" (anonymous, date unknown).

3 A SIMPLE FRAMEWORK FOR VERIFICATION AND VALIDATION

The simplest and the most obvious techniques are usually the best. First, they are more likely to be understood and used. Second, they are often overlooked, but when used often uncover model defects.

Here is a suggested framework for verification and validation:

1. Test the model for face validity.
2. Test the model over a range of input parameters.
3. Where applicable, compare model predictions to past performance of the actual system or to a baseline model representing an existing system. When designing a new system, compare implemented model behavior to assumptions and specifications.

First, test the model for face validity. For a given scenario, examine all the model's output measures of performance and ask "are they reasonable?". Examine as many outputs as possible. (See Section 4 for a detailed discussion of relevant output measures.)

Second, run the model over the widest range of input parameters that are likely to be varied during the course of experimentation. Consider this a "stress test". Examine trends in common measures of performance, such as throughput, when some input is changed; usually at least the direction if not the magnitude of change is known. Look for outliers in system performance – outputs that are way out of line with trends or expectations. Examine the runs that produced those results in detail.

When some setting of a model's inputs allows the model to match an existing system, then (and only then) a scientific validation is at least theoretically possible, and becomes possible in practice if the right data can be collected. In this situation, if at all possible:

1. Collect input data and corresponding system performance measures for some period of time.
2. Run the model with the given input data.
3. Compare model performance to real-world performance over the given period of time.
4. If one set of data (input and outputs) are available, compare on a reasonableness basis. If two or more sets are available, use statistical techniques such as confidence intervals to compare the difference in model performance and real-system performance.

When designing a new system, a completely scientific validation is not possible simply because a real-system does not exist as a basis for comparison. In this situation, it is essential that system designers examine and verify model behavior at the micro-level. This includes how the model responds to extreme as well as normal situations. Since many questions that may be addressed during experimentation may depend on how well a model incorporates cause and effect, it is essential that the system design as implemented in the model be verified at this level. This includes not only its physical design, but at least as important, any control schemes, operational rules and policies, and labor and other human interventions.

4 MEASURES OF PERFORMANCE FOR V & V

It is often stated that a model should be built for a specified purpose or set of objectives, and even though validated for the original objectives, the model may or may not be valid for another purpose or set of objectives. My view differs a bit. It is my belief that a model, once its validity is proven to everyone's satisfaction for one purpose, should be valid for any other purpose *within its scope and level of detail*.

If the model, because of its scope and level of detail, can measure what's needed, then a validated model should be valid when the purpose or objective changes. A model that is valid should be valid for any purpose or objective that can be met by a measure of performance which the model can capture. A model may not be capable of addressing a new objective, but that is most likely to be because of the original model's limited scope or lack of detail in some area, not because it was developed with another purpose in mind.

In the initial phase of any modeling project, the model developer, decision makers, systems experts and other involved parties need to agree on, among other things:

1. The objectives of the study.
2. The questions to be addressed in order to meet the objectives.

3. The measures of performance that will be used to answer the questions of interest.
4. The model's scope and level of detail for various subsystems, keeping the objectives and questions in mind so that the model will be capable of addressing those objectives and answering those questions

Typical measures of performance for industrial models include the primary measures such as throughput, system cycle or response time, and work in process. In addition, a number of secondary or explanatory measures may be of interest, such as resource utilization, size of local buffers, and throughputs for subsystems or particular part types.

It is usually not possible to identify ahead of time all possible output measures of performance that might potentially be of interest. Here's a typical scenario during the experimentation phase. Example: A model configuration that was expected to perform well in fact does not perform well. The typical primary measure mentioned earlier identify a departure from expectations. Explanations are required but are not immediately obvious or forthcoming; the cause or causes are hidden or veiled. After investigation, watching the animation, examining time lines of key variables, and some thought, an hypothesis is developed for the reason for the departure from expectation. Someone says, "Prove it!". To prove it, it turns out that some new measures of subsystem performance are needed, so they are added to the model, runs are made and the hypothesis is confirmed or contradicted by the new measures. If contradicted, the process repeats. During the process, insights are gained and our understanding of system dynamics increases.

Some points learned from this example: Primary measures tell us what happened, but often secondary or auxiliary measures are needed to confirm insights and provide explanations of why they happened. This leads to a second point: secondary measures are often the most useful in detecting model shortcomings and invalid logic. A model is not valid until all of its subsystems are valid.

In addition, global measures of performance often mask serious errors in a model. It is essential to verify and validate a wide range of model output measures. We recommend that "local" as well as global measures be captured and examined. Depending on the model, "local" might mean statistics on individual resources, buffers and subsystems or statistics on individual part types.

Example 6: A manufacturer of printed circuit boards asked our consulting group to review a model that had been developed by a previous employee and had not been used for two years. We found that the model had been set up to run for 31 days and printed a custom report at the end of 31 days on overall production throughput for broad classes of boards but not for individual board types. Close examination of the inputs revealed that one board type representing roughly 1% of desired total production took at least 30 days to get to its last production step. (This was due to longer testing times than other board types as well as to batching requirements for certain heat treatments.) Therefore in runs of 31 days, few or sometimes none of these boards completed production. Looking at overall global measures of performance obscured this fact; examining detailed statistics on individual part types revealed this (and other) modeling and potential experimentation errors.

5 SOME SUGGESTED TECHNIQUES

Here we provide a list of V&V techniques that may prove useful in certain types of models:

- Force rare events and extreme cases so that model behavior responding to these events can be tested.
- Identify output values that indicate a modeling error or suspicious behavior.
- Identify internal system conditions that indicate a modeling error. Write "`if …`" statements to test for these conditions and report them to a message or log file.
- Make lots of runs. Before beginning the formal experimentation, do extensive model testing by making runs over a wide range of input parameter settings. Conduct a "trend" analysis to see if typical outputs (e.g., throughput) at least go in the expected direction (up or down) when some input is increased (e.g., machine speeds or number of workers).
- In vehicle models (including models in which operators walk from task to task), check for vehicle lockup (or inordinate human "idle" time). Vehicle lockup can occur due to faulty control logic.
- Use timelines to view current statistics (not summary statistics). These include items such as current work-in-process and other queue sizes, resources being used, and number of active vehicles. Such timelines can quickly reveal major modeling errors that lead to some resource not being used for a lengthy period of time.
- Use timelines to view current work-in-process in all major subsystems. Count entities into and out of each subsystem to be sure all are accounted for. (I have seen models where entities get "lost" in the model.)
- Use the animation intelligently. It is excellent for verifying model behavior on the micro-level over short time frames.
- Examine a wide variety of output measures
  - primary measures such as throughput
  - more than the primary measures
    - "local" measures for individual resources (machines or workstations or vehicles)

  - measures for each type of "object" (e.g., by part type, not just overall parts)
- examine for reasonableness, which means to detect those that are clearly out of line.

6 SUMMARY

Verification and validation are essential phases in the model development process for any simulation project. Use of the simple techniques outlined in this paper can assist you in avoiding major and serious modeling errors.

REFERENCES

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2001. *Discrete-Event System Simulation*, third edition, Prentice-Hall, Upper Saddle River, N. J.

Law, A. M. and W. D. Kelton. 2000. *Simulation Modeling and Analysis*, third edition, McGraw-Hill, New York.

Law, A. M. and M. G. McComas. 2001. How to Build Valid and Credible Simulation Models. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 22-29. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Petroski, Henry. 1992. *To Engineer Is Human: The Role of Failure in Successful Design.* New York: St. Martin's Press, 1985. London: Macmillan, 1986. Tokyo: Kajima Institute Press, 1988 (Japanese translation). New York: Vintage Books (paperback).

Sargent, R. G. 2001. Some Approaches and Paradigms for Verifying and Validating Simulation Models. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 106-114. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Schmeiser, B. W. 2001. Some Myths and Common Errors in Simulation Experiments. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 39-46. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Schriber, T. J., S. Cox, J. O. Henriksen, P. Lorenz, J. Reitman, and I. Ståhl. 2001. GPSS Turns 40: Selected Perspectives. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 565-576. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

AUTHOR BIOGRAPHY

JOHN S. CARSON II is the Consulting Technical Manager for the AutoMod Group of Brooks-PRI Automation, Planning & Logistics Solutions division. He has over 25 years experience in simulation in a wide range of application areas, including manufacturing, material handling, warehousing and distribution, transportation, ports and shipping, and health care systems. With the AutoMod Group for 8 years, previously he was President and founder of Carson/Banks & Associates, an independent simulation consulting firm. Before that, he taught simulation and operations research at Georgia Tech and was an independent consultant. He also taught at the University of Florida and the University of Wisconsin. He is the co-author of two university level textbooks including the widely used Discrete-Event Systems Simulation (third edition, 2001). He holds a Ph.D. in Industrial Engineering and Operations Research from the University of Wisconsin-Madison, and is a senior member of IIE and INFORMS. His e-mail address is <John.Carson@brooks-pri.com>.