## 1. INTRODUCTION

Simulation is the process of constructing a model of a system which contains a problem and conducting experiments with the model on a computer for a specific purpose of experimentation to solve the problem. Credibility of simulation results not only depends on model correctness, but also is significantly influenced by accurate formulation of the problem. Therefore, validation, verification, and testing (VV&T) techniques must be employed throughout the life cycle of a simulation study starting with problem formulation and culminating with presentation of simulation results.

A *model* is a representation and an abstraction of anything such as a system, concept, problem, or phenomena. It can have inputs, parameters, and outputs as illustrated in Figure 1. The term "system" is used to refer to the entity that contains the problem to be solved.

*Model Validation* is substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives. Model validation deals with building the *right* model. It is conducted by running the model under the "same" input conditions that drive the system and by comparing model behavior with the system behavior. The comparison of model and system behaviors should *not* be made one output variable at a time, i.e., $O_1^m$ versus $O_1^s$, $O_2^m$ versus $O_2^s$, etc. A multivariate comparison should be carried out to incorporate the correlations among the output variables.

*Model Verification* is substantiating that the model is transformed from one form into another, as intended, with sufficient accuracy. Model verification deals with building the model *right*. The accuracy of transforming a problem formulation into a model specification or the accuracy of converting a model representation in micro flowchart into an executable computer program is evaluated in model verification.

*Model Testing* is demonstrating that inaccuracies exist or revealing the existence of errors in the model. In model testing, we subject the model to test data or test cases to see if it functions properly. "Test failed" implies the failure of the model, not the test. Testing is conducted to perform validation and verification. Some tests are devised to evaluate the behavioral accuracy (i.e., validity) of the model, and some tests are intended to judge the accuracy of model transformation from one form into another (verification). Therefore, we commonly refer to the whole process as *model VV&T*.

Model VV&T is employed to prevent the occurrence of three major types of errors in conducting simulation studies [Balci 1990]: *Type I Error* is the error of rejecting the model credibility when in fact the model is sufficiently credible. *Type II Error* is the error of accepting the model credibility when in fact the model is *not* sufficiently credible. *Type III Error* is the error of solving the wrong problem. Probability of committing the Type I Error is called *Model Builder's Risk* and probability of committing the Type II Error is called Model User's Risk. Committing the Type I error increases the cost of model

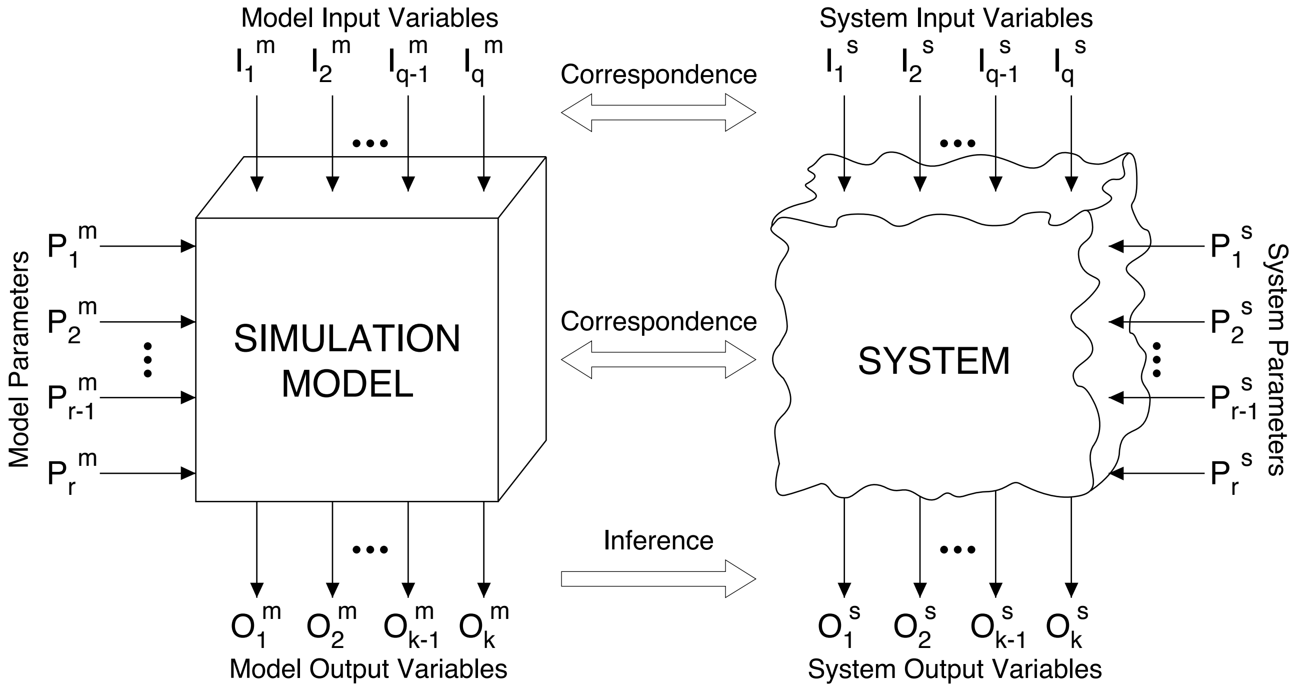


**Figure 1.** Model and System Characteristics

development. The consequences of committing the Type II and Type III errors may be catastrophic. Therefore, a cost-risk analysis should be conducted in those cases where data can be collected from the system under study [Balci and Sargent 1981].

Significant differences exist between simulation software engineering and other types of software engineering. First, simulation software engineering corresponds to simulation modeling and as such the art of modeling should be applied. Second, the results are obtained by experimenting with the simulation program (experimental model) as opposed to just executing it once like other types of programs. Third, the results are descriptive and must be carefully interpreted to come up with a solution to the problem. Fourth, software requirements specification corresponds to problem (system) description. Validation is conducted by comparing the model (computer program) with the system description as opposed to with the requirements specification. In spite of these differences, all software VV&T techniques are directly applicable for simulation models.

Every organization conducting a substantial simulation study should have a department or group called Simulation Quality Assurance (SQA). The SQA group is responsible for total quality management and closely works with the simulation project managers in planning, preparing test cases, and administering some of the VV&T activities throughout the simulation study. The SQA is a managerial approach which is critically essential for the success of a simulation study. Ören [1981, 1986, 1987] presents concepts, criteria, and paradigms which can be used in establishing an SQA program within an organization.

The purpose of this paper is to survey current software VV&T techniques and current model VV&T techniques and describe how they can all be applied throughout the life cycle of a simulation study. Section 2 presents the life cycle of a simulation study and provides guidelines for conducting its ten processes. The VV&T techniques are briefly described under a taxonomy in Section 3. Section 4 describes the credibility assessment stages of the life cycle and shows the applicability of the VV&T techniques for each stage. Concluding remarks and research directions are given in Section 5. A glossary is provided in Section 6 to explicitly define important terms and VV&T techniques.

## 2. THE LIFE CYCLE OF A SIMULATION STUDY

The life cycle of a simulation study is presented in Figure 2 [Balci 1990; Nance 1981]. The phases are shown by shaded oval symbols. The dashed arrows describe the processes which relate the phases to each other. The solid arrows refer to the credibility assessment stages. Banks *et al.* [1987] and Knepell and Arangno [1993] review other modeling processes for developing simulations.

The life cycle should not be interpreted as strictly sequential. The sequential representation of the dashed arrows is intended to show the direction of development throughout the life cycle. The life cycle is iterative in nature and reverse transitions are expected. Every phase of the life cycle has an associated VV&T activity. Deficiencies identified by a VV&T activity may necessitate returning to an earlier process and starting all over again.

The VV&T is not a phase or step in the life cycle, but a continuous activity throughout the entire life cycle. Conducting the VV&T for the first time in the life cycle when the experimental model is complete is analogous to the teacher who gives only a final examination [Hetzel 1984]. No opportunity is provided throughout the semester to notify the student that he or she has serious deficiencies. Severe problems may go undetected until it is too late to do anything but fail the student. Frequent tests and homeworks throughout the semester are intended to inform the students about their deficiencies so that they can study more to improve their knowledge as the course progresses.

The situation in the VV&T is exactly analogous. The VV&T activities throughout the entire life cycle are intended to reveal any quality deficiencies that might be present as the simulation study progresses from the communication of the problem until the implementation of the simulation results. This allows us to identify and rectify quality deficiencies during the life cycle phase in which they occur.

The ten processes of the life cycle are shown by the dashed arrows in Figure 2. Although each process is executed in the order indicated by the dashed arrows, an error identified may necessitate returning to an earlier process and starting all over again. Some guidelines are provided below for each of the ten processes.

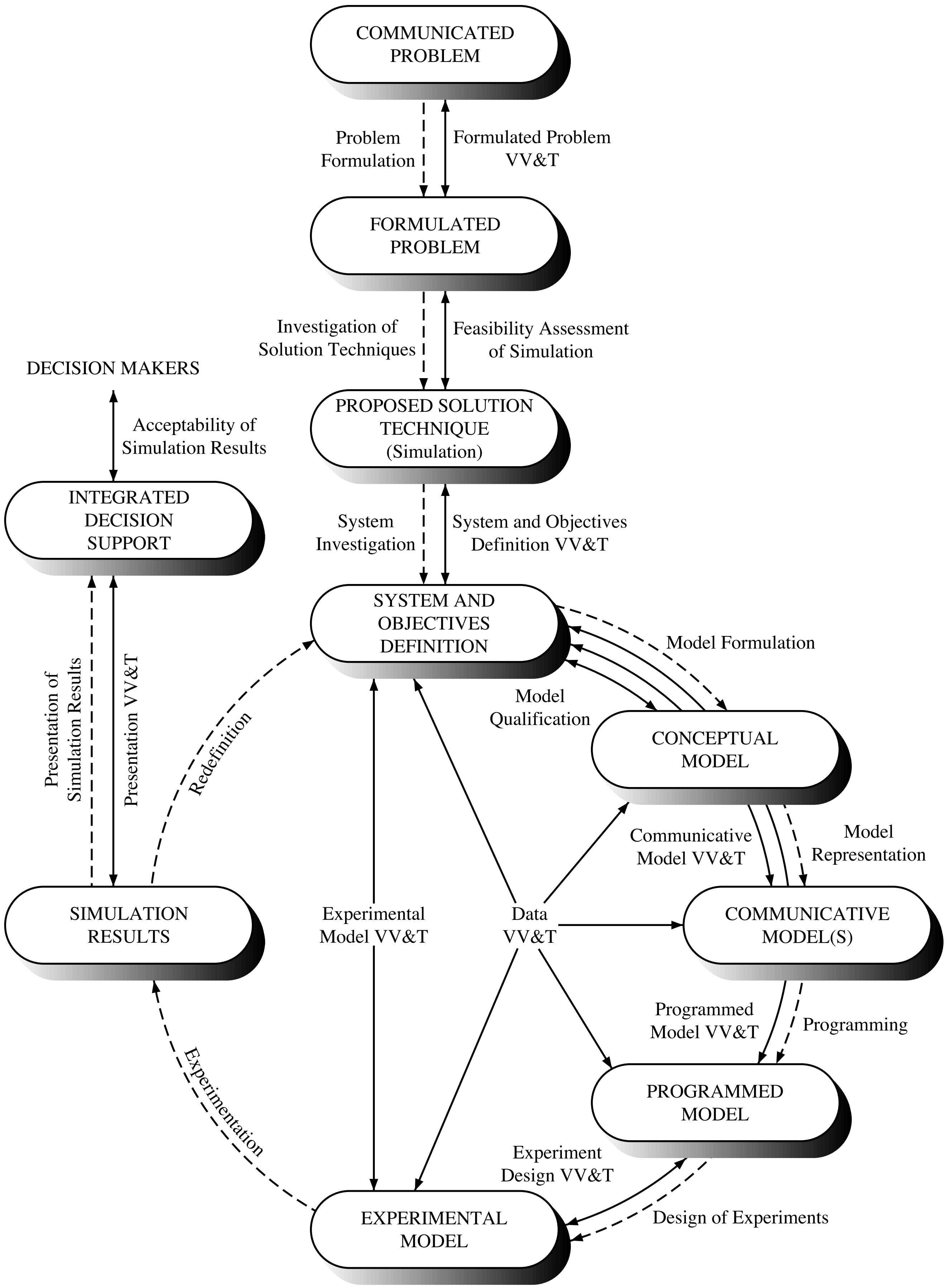


**Figure 2.** The Life Cycle of a Simulation Study

### 2.1 Problem Formulation

When a problem is recognized, a decision maker (a client or sponsor group) initiates a study by communicating the problem to an analyst (a problem-solver, or a consultant/research group). The communicated problem is rarely clear, specific, or organized. Hence, an essential study to formulate the *actual* problem must follow. *Problem Formulation* (problem structuring or problem definition) is the process by which the initially communicated problem is translated into a formulated problem sufficiently well defined to enable specific research action [Woolley and Pidd 1981].

Balci and Nance [1985] present a high-level procedure that: (1) guides the analyst during problem formulation, (2) structures the formulated problem VV&T, and (3) seeks to increase the likelihood that the study results are utilized by decision makers.

### 2.2 Investigation of Solution Techniques

All alternative techniques that can be used in solving the formulated problem should be identified. A technique whose solution is estimated to be too costly or is judged to be not sufficiently beneficial with respect to the study objectives should be disregarded. Among the qualified ones, the technique with the highest expected benefits/cost ratio should be selected.

The statement "when all else fails, use simulation" is misleading if not invalid. The question is not to bring <u>a</u> solution to the problem, but to bring a sufficiently credible one which will be accepted and used by the decision maker(s). A technique other than simulation may provide a less costly solution, but it may not be as useful.

Sometimes, the communicated problem is formulated under the influence of a solution technique in mind. Occasionally, simulation is chosen without considering any other technique just because it is the only one the analyst(s) can handle. Skipping the investigation process may result in unnecessarily expensive solutions, sometimes to the wrong problems.

As a result of the investigation process, we assume that simulation is chosen as the most appropriate solution technique. At this point, the simulation project team should be activated and be made responsible for the formulated problem VV&T and feasibility assessment of simulation before proceeding in the life cycle.

### 2.3 System Investigation

Characteristics of the system that contains the formulated problem should be investigated for consideration in system definition and modeling. Shannon [1975] identifies six major system characteristics: (1) change, (2) environment, (3) counterintuitive behavior, (4) drift to low performance, (5) interdependency, and (6) organization. Each characteristic should be examined with respect to the study objectives that are identified with the formulation of the problem.

In simulation, we mostly deal with stochastic and dynamic real systems that *change* over a period of time. How often and how much the system will change during the course of a simulation study should be estimated so that the model representation can be updated accordingly. Changes in the system may also change the study objectives.

A system's *environment* consists of all input variables that can significantly affect its state. The input variables are identified by assessing the significance of their influence on the system's state with regard to the study objectives. For example, for a traffic intersection system, the interarrival time of vehicles can be identified as an input variable making up the environment, whereas pedestrian arrivals may be omitted due to their negligible effect on the system's state. Underestimating the influence of an input variable may result in inaccurate environment definition.

Some complex systems may show *counterintuitive behavior* which we should try to identify for consideration in defining the system. However, this is not an easy task, especially for those systems containing many subjective elements (e.g., social systems). Cause and effect are often not closely related in time or space. Symptoms may appear long after the primary causes [Shannon 1975]. To be able to identify counterintuitive behavior, it is essential that the simulation project employs people who have expert knowledge about the system under study.

A system may show a *drift to low performance* due to the deterioration of its components (e.g., machines in a manufacturing system) over a period of time. If this characteristic exists, it should be incorporated within the model representation especially if the model's intended use is forecasting.

Before we start abstracting the real system for the purpose of modeling, we should examine the *interdependency* and *organization* characteristics of the system. In a complex stochastic system, many activities or events take place simultaneously and influence each other. The system complexity can be overcome by way of decomposing the system into subsystems and subsystems into other subsystems. This decomposition can be carried out by examining how system elements or components are organized.

Once the system is decomposed into subsystems the complexity of which is manageable and the system characteristics are documented, model formulation process can be started following the system and objectives definition VV&T.

## 2.4 Model Formulation

Model formulation is the process by which a conceptual model is envisioned to represent the system under study. The *Conceptual Model* is the model which is formulated in the mind of the modeler [Nance 1981]. Model formulation and model representation constitute the process of model design.

*Input data analysis and modeling* [Law and Kelton 1991] is a subprocess of Model Formulation and is conducted with respect to the way the model is driven. Simulation models are classified as self-driven or trace-driven. A *self-driven* (distribution-driven or probabilistic) simulation model is the one which is driven by input values obtained via sampling from probability distributions using random numbers. A *trace-driven* (or retrospective) simulation model, on the other hand, is driven by input sequences derived from trace data obtained through measurement of the real system.

Under some study objectives (e.g., evaluation, comparison, determination of functional relations) and for model validation, input data model(s) are built to represent the system's input process. In a self-driven simulation (e.g., of a computer system), we collect data on an input random variable (e.g., inter-arrival time of jobs), identify the distribution, estimate its parameters, and conclude upon a probability distribution as the input data model to sample from in driving the simulation model [Law and Kelton 1991]. In a trace-driven simulation, we trace the system (e.g., using hardware and software monitors) and utilize the refined trace data as the input data model to use in driving the simulation model.

## 2.5 Model Representation

This is the process of translating the conceptual model into a communicative model. A *Communicative Model* is "a model representation which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human" [Nance 1981]. A communicative model (i.e., a simulation model design) may be represented in any of the following forms: (1) structured, computer-assisted graphs, (2) flowcharts, (3) structured English and pseudocode, (4) entity-cycle (or activity-cycle) diagrams, (5) condition specification [Overstreet and Nance 1985], and (6) more than a dozen diagramming techniques described in [Martin and McClure 1985].

Several communicative models may be developed; one in the form of Structured English intended for nontechnical people, another in the form of a micro flowchart intended for a programmer. Different representation forms may also be integrated in a stratified manner.

The representation forms should be selected based upon: (1) their applicability for describing the system under study, (2) the technical background of the people to whom the model is to be communicated, (3) how much they lend themselves to formal analysis and verification, (4) their support for model documentation, (5) their maintainability, and (6) their automated translatability into a programmed model.

## 2.6 Programming

Translation of the communicative model into a programmed model constitutes the process of programming. A *Programmed Model* is an executable simulation model representation in a simulation programming language (e.g., GPSS, SIMAN, SIMSCRIPT, SIMULA, SLAM, etc.) or in a high-level programming language (e.g., C, Fortran, Pascal, etc.) that do not incorporate an experiment design. There is an abundance of literature on simulation programming languages. Balci [1988] describes how to conduct the programming process in high-level languages.

### 2.7 Design of Experiments

This is the process of formulating a plan to gather the desired information at minimal cost and to enable the analyst to draw valid inferences [Shannon 1975]. An *Experimental Model* is the programmed model incorporating an executable description of operations presented in such a plan.

A variety of techniques are available for the design of experiments. *Response-surface methodologies* can be used to find the optimal combination of parameter values which maximize or minimize the value of a response variable [Law and Kelton 1991]. *Factorial designs* can be employed to determine the effect of various input variables on a response variable [Fishman 1978]. *Variance reduction techniques* can be implemented to obtain greater statistical accuracy for the same amount of simulation [Law and Kelton 1991]. *Ranking and selection techniques* can be utilized for comparing alternative systems [Law and Kelton 1991; Banks and Carson 1984]. Several methods (e.g., replication, batch means, regenerative) can be used for statistical analysis of simulation output data.

### 2.8 Experimentation

This is the process of experimenting with the simulation model for a specific purpose. Some purposes of experimentation are [Shannon 1975]: (1) comparison of different operating policies, (2) evaluation of system behavior, (3) sensitivity analysis, (4) forecasting, (5) optimization, and (6) determination of functional relations. The process of experimentation produces the Simulation Results.

### 2.9 Redefinition

This is the process of: (1) updating the experimental model so that it represents the current form of the system, (2) altering it for obtaining another set of results, (3) changing it for the purpose of maintenance, (4) modifying it for other use(s), or (5) redefining a new system to model for studying an alternative solution to the problem.

### 2.10 Presentation of Simulation Results

In this process, simulation results are interpreted and presented to the decision makers for their acceptance and implementation. Since all simulation models are descriptive, concluding upon a solution to the problem requires rigorous analysis and interpretation of the results.

The presentation should be made with respect to the intended use of the model. If the model is used in a “what if” environment, the results should be integrated to support the decision maker in the decision-making process. Complex simulation results may also necessitate such an integration. The report documenting the study and its results together with its presentation also constitutes a form of supporting the decision maker.

## 3. VALIDATION, VERIFICATION, AND TESTING TECHNIQUES

Figure 3 shows a taxonomy which categorizes the VV&T techniques into six distinct credibility assessment perspectives: informal, static, dynamic, symbolic, constraint, and formal. The level of mathematical formality of each category increases from very informal on the far left to very formal on the far right. Likewise, the complexity also increases as the category becomes more formal [Whitner and Balci 1989].

It should be noted that some of the categories presented in Figure 3 possess similar characteristics and in fact have techniques which overlap from one category to another. However, a distinct difference between each classification exists, as will be evident in the discussion of each in this section.

### 3.1 Informal VV&T Techniques

Informal techniques are among the most commonly used ones. They are called informal because the tools and approaches used rely heavily on human reasoning and subjectivity without stringent mathematical formalism. The "informal" label does not imply any lack of structure or formal guidelines for the use of the techniques.

#### *3.1.1 Audit*

The audit is undertaken by a single person to investigate how adequately the simulation study is conducted with respect to established practices, standards, and guidelines. The audit also seeks to establish traceability within the simulation study. When an error is identified, it should be traceable to its source via its audit trail. Auditing is carried out on a periodic basis through a mixture of meetings, observations, and examinations. [Hollocker 1987]

#### *3.1.2 Desk Checking*

Desk Checking is the process of thoroughly examining one's work to ensure correctness, completeness, consistency, and unambiguity. It is considered to be the very first step in VV&T and is particularly useful for the early stages of development. To be effective, Desk Checking should be conducted carefully and thoroughly preferably by another person since it is usually difficult to see one's own errors. [Adrion et al. 1982]

#### *3.1.3 Face Validation*

The project team members, potential users of the model, people knowledgeable about the system under study, based on their estimates and intuition, subjectively compare model and system behaviors to judge whether the model and its results are reasonable. Face Validation is useful as a preliminary approach to validation. [Hermann 1967]

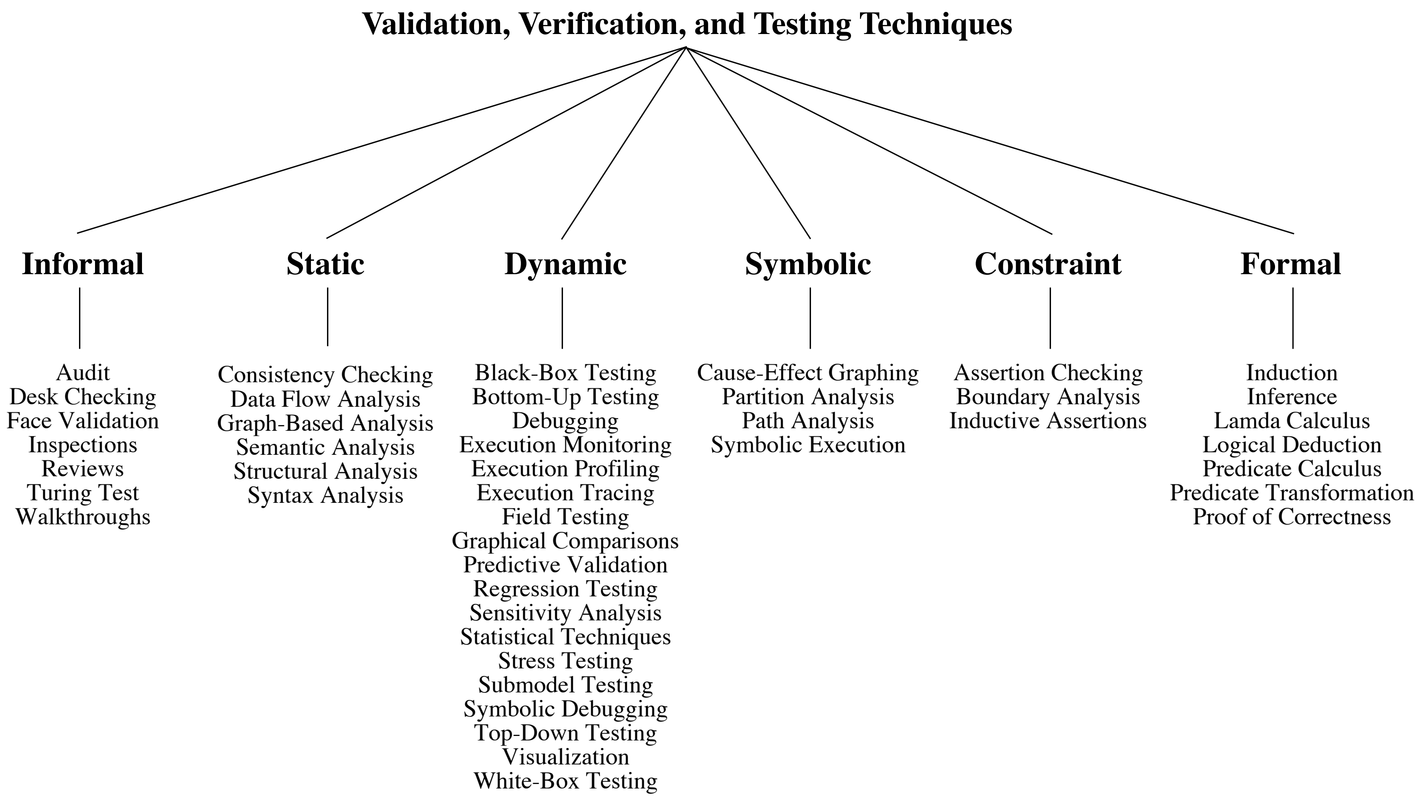


**Figure 3.** A Taxonomy of Validation, Verification, and Testing Techniques

*3.1.4 Inspections*

Inspections are conducted by a team of four to six members. For example, in the case of a design inspection, the team consists of: (1) Moderator: manages the inspection team and provides leadership; (2) Reader: narrates the model design (communicative model) and leads the team through it; (3) Recorder: produces a written report of detected faults; (4) Designer: is the creator of the model design; (5) Implementer: translates the model design into code (programmed model); and (6) Tester: SQA group representative.

An inspection goes through five distinct phases: overview, preparation, inspection, rework, and follow-up [Schach 1993]. In phase I, the designer gives an overview of the (sub)model design to be inspected. The (sub)model characteristics such as purpose, logic, and interfaces are introduced and related documentation is distributed to all participants to study. In phase II, the team members prepare individually for the inspection by examining the documents in detail. The moderator arranges the inspection meeting with an established agenda and chairs it in phase III. The reader narrates the (sub)model design documentation and leads the team through it. The inspection team is aided by a checklist of queries during the fault finding process. The objective is to find and document the faults, not to correct them. The recorder prepares a report of detected faults immediately after the meeting. Phase IV is the rework in which the designer resolves all faults and problems specified in the written report. In the final phase, the moderator ensures that all faults and problems have been resolved satisfactorily. All changes must be examined carefully to ensure that no new errors have been introduced as a result of a fix.

A disadvantage of the inspection technique is that, like the walkthrough, it might be used for performance appraisal of the development team. Major differences exist between inspections and walkthroughs. An inspection is a five-step process, but walkthroughs consist of only two steps. The inspection team uses the checklist approach for uncovering errors. The procedure used in each phase of the inspection technique is formalized. The inspection process takes much longer than a walkthrough; however, the extra time is justified because an inspection is a powerful and cost-effective way of detecting faults early in the model development life cycle. [Ackerman et al. 1983; Dobbins 1987; Knight and Myers 1993; Schach 1993]

#### *3.1.5 Reviews*

The review is conducted in a similar manner as the inspection and walkthrough except in the way the team members are selected. The review team also involves managers. The review is intended to give management and study sponsors evidence that the development process is being carried out according to stated study objectives and evaluate the model in light of development standards, guidelines, and specifications. As such, the review is a higher level technique than the inspection and walkthrough.

Each review team member examines the model documentation prior to the review. The team then meets to evaluate the model relative to specifications and standards, recording defects and deficiencies. The review team may be given a set of indicators to measure such as: (1) appropriateness of the definition of system and study objectives, (2) adequacy of all underlying assumptions, (3) adherence to standards, (4) modeling methodology used, (5) model representation quality, (4) model structuredness, (5) model consistency, (6) model completeness, and (7) documentation. The result of the review is a document portraying the events of the meeting, deficiencies identified, and review team recommendations. Appropriate action may then be taken to correct any deficiencies.

As opposed to inspections and walkthroughs, which concentrate on correctness assessment, reviews seek to ascertain that tolerable levels of quality are being attained. The review team is more concerned with model design deficiencies and deviations from stated model development policy than it is with the intricate line-by-line details of the implementation. This does not imply that the review team is not concerned with discovering technical flaws in the model, only that the review process is oriented towards the early stages of the model development life cycle. [Hollocker 1987; Whitner and Balci 1989]

### 3.1.6 *Turing Test*

Turing Test is based upon the expert knowledge of people about the system under study. These people are presented with two sets of output data obtained, one from the model and one from the system, under the same input conditions. Without identifying which one is which, the people are asked to differentiate between the two. If they succeed, they are asked how they were able to do it. Their response provides valuable feedback for correcting model representation. If they cannot differentiate, our confidence in model validity is increased. [Schruben 1980; Turing 1963; Van Horn 1971]

### 3.1.7 *Walkthroughs*

Walkthroughs are conducted by a team composed of a coordinator, model developer, and three to six other members. Except the model developer, all other members should not be directly involved in the development effort. A typical structured walkthrough team consists of: (1) Coordinator: most often is the SQA group representative who organizes, moderates, and follows up the walkthrough activities; (2) Presenter: most often is the model developer; (3) Scribe: documents the events of the walkthrough meetings; (4) Maintenance Oracle: considers long-term implications; (5) Standards Bearer: concerned with adherence to standards; (6) Client Representative: reflects the needs and concerns of the client; and (7) Other reviewers such as simulation project manager and auditors.

The main thrust of the walkthrough technique is to detect and document faults; it is *not* performance appraisal of the development team. This point must be made clear to everyone involved so that full cooperation is achieved in discovering errors.

The coordinator schedules the walkthrough meeting, distributes the walkthrough material to all participants well in advance of the meeting in order to allow for careful preparation, and chairs the meeting. During the meeting, the presenter walks the other members through the walkthrough documents. The coordinator encourages questions and discussion so as to uncover any faults. [Adrion et al. 1982; Deutsch 1982; Myers 1978, 1979; Yourdon 1985]

## 3.2 Static VV&T Techniques

Static VV&T techniques are concerned with accuracy assessment on the basis of characteristics of the static model source code. Static techniques do not require machine execution of the model, but mental execution may be used. The techniques are very popular and widely used, with many automated tools available to assist the VV&T. The simulation language compiler is itself a static VV&T tool.

Static VV&T techniques can obtain a variety of information about the structure of the model, coding techniques and practices employed, data and control flow within the model, syntactical accuracy, and internal as well as global consistency and completeness of implementation [Whitner and Balci 1989].

### *3.2.1 Consistency Checking*

Consistency checking deals with substantiating that: (a) the model representation does not contain contradictions, (b) the cosmetic style with which language elements (e.g., naming conventions, use of upper, lower, and mixed case, etc.) are applied is used consistently, and (c) the data elements are manipulated uniformly (e.g., data assignment to variables, data use within computations, data passing among submodels, data representation and use during model input and output).

### *3.2.2 Data Flow Analysis*

Data flow analysis is used to assess model accuracy with respect to the use of model variables. This assessment is classified according to the definition, referencing, and unreferencing of variables [Adrion et al. 1982], i.e., when variable space is allocated, accessed, and deallocated. A data flow graph is constructed to aid in the data flow analysis. The nodes of the graph represent statements and corresponding variables. The edges represent control flow.

Data flow analysis can be used to detect undefined or unreferenced variables (much as in static analysis) and, when aided by model instrumentation, can track minimum and maximum variable values, data dependencies, and data transformations during model execution. It is also useful in detecting inconsistencies in data structure declaration and improper linkages among submodels. [Allen and Cocke 1976; Whitner and Balci 1989]

### *3.2.3 Graph-Based Analysis*

Data-flow and control-flow are both graph-based analysis techniques which are similar in many ways [Adrion et al. 1982]. Data-flow analysis is described in Section 3.2.2. In control-flow analysis, a node of the model graph usually represents a logical junction where the flow of control changes, while an edge represents towards which junction it changes. This technique examines sequences of control transfers and is useful for identifying incorrect or inefficient constructs within model representation.

Nance and Overstreet [1987] proposed several diagnostics which are based on analysis of graphs constructed from a particular form of model specification called Condition Specification [Overstreet and Nance 1985; Moose and Nance 1989]. The diagnostic assistance is categorized into three: (1) analytical—determination of the existence of a property, (2) comparative—measures of differences among multiple model representations, and (3) informative—characteristics extracted or derived from model representations. Action cluster attribute graph, action cluster incidence graph, and run-time graph constitute the basis for the diagnosis.

The analytical diagnosis is conducted by measuring the following indicators: attribute utilization, attribute initialization, action cluster completeness, attribute consistency, connectedness, accessibility, out-complete, and revision consistency. The comparative diagnosis is done by measuring attribute cohesion, action cluster cohesion, and complexity. The following indicators are measured for the informative diagnosis: attribute classification, precedence structure, decomposition, and run-time graph [Nance and Overstreet 1987].

### *3.2.4 Semantic Analysis*

Semantic analysis is conducted by the simulation programming language compiler and attempts to determine the modeler's intent in writing the code. The compiler informs the modeler about what is specified in the source code so that the modeler can verify that the true intent is accurately reflected.

The compiler generates a wealth of information to help the modeler determine if the true intent is accurately translated into the executable code: (1) *Symbol Tables* which describe: the elements or symbols that are manipulated in the model, function declarations, type and variable declarations, scoping relationships, interfaces, dependencies, etc.; (2) *Cross-reference Tables* which describe: called versus calling submodels (where each data element is declared, referenced and altered), duplicate data declarations (how often and where occurring), and unreferenced source code; (3) *Subroutine Interface Tables* which describe the actual interfaces of the caller and the called; (4) *Maps* which relate the generated runtime code to the original source code; and (5) *"Pretty Printers"* or *Source Code Formatters* which provide: reformatted source listing on the basis of its syntax and semantics, clean pagination, highlighting of data elements, and marking of nested control structures.

### *3.2.5 Structural Analysis*

Structural analysis is used to examine the model structure and to determine if it adheres to structured principles. It is conducted by constructing a control flow graph of the model structure and examining the graph for anomalies, such as multiple entry and exit points, excessive levels of nesting within a structure, and questionable practices such as the use of unconditional branches (i.e., GOTOs).

Yücesan and Jacobson [1992] and Jacobson and Yücesan [1993] apply the theory of computational complexity and show that the problem of verifying structural properties of simulation models are intractable. They illustrate that modeling issues such as accessibility of states, ordering of events, ambiguity of model specifications, and execution stalling are NP-complete decision problems.

### *3.2.6 Syntax Analysis*

Syntax analysis is carried on by the simulation programming language compiler to assure that the mechanics of the language are applied correctly.

## 3.3 Dynamic VV&T Techniques

Dynamic VV&T techniques require model execution and are intended for evaluating the model based on its execution behavior. Most dynamic VV&T techniques require model instrumentation.

The insertion of additional code (probes) into the executable model for the purpose of collecting information about model behavior during execution is called *model instrumentation*. Probe locations are determined manually or automatically based on static analysis of model structure. Automated instrumentation is accomplished by a preprocessor which analyzes the model static structure (usually via graph-based analysis) and inserts probes at appropriate places.

The simplest probe type is a counter and is illustrated in Figure 4. In this example, computer jobs arrive according to a Poisson process. A fundamental assumption underlying a Poisson arrival process is that the interarrival times must be nonzero. Therefore, to find out what percentage of time this assumption is violated, the model in Figure 4 is instrumented by inserting Blocks 3-6. Running the model for 50,000 jobs reveals that the percentage of violation (100*COUNT/Total number of entries into Block 4) is 18% (see [Schriber 1974, p. 165] for an explanation). To decrease this unacceptable frequency of violation, the time unit can be changed to 10 milliseconds; thus, changing 5 and 4 in the INITIAL Block to 50 and 40 reduces the percentage of violation to 2%. If 100 milliseconds were chosen as the time unit, the frequency of violation would only be 0.19%. Consequently, the higher the value of MIAT the lower the frequency of violation.

Dynamic VV&T techniques are usually applied using the following three steps. In Step 1, the programmed or experimental model is instrumented. In Step 2, the instrumented model is executed, and in Step 3, the model output is analyzed and dynamic model behavior is evaluated.

### *3.3.1 Black-Box Testing*

Black-box testing, also called functional testing, is used to assess the accuracy of model input-output transformation. It is applied by feeding inputs (test data) to the model and evaluating the corresponding outputs. The concern is how accurately the model transforms a given set of input data into a set of output data.

It is virtually impossible to test all input-output transformation paths for a reasonably large and complex simulation model since the number of those paths could be in millions. Therefore, the objective of black-box testing is to increase our confidence in model input-output transformation accuracy as much as possible rather than trying to claim absolute correctness.

Generation of test data is a crucially important but a very difficult task. The law of large numbers does not apply here. Successfully testing the model under 1,000 input values (test data) does not imply high confidence in model input-output transformation accuracy just because 1,000 is a large number.

```
***********************************************************************
*  This is a GPSS/H programmed model of a single Central Processing Unit
*  (CPU) serving computer jobs arriving according to a Poisson process
*  with First-Come First-Served (FCFS) queue discipline. Job processing
*  times follow an exponential probability distribution.
*
*       MIAT       =  Mean InterArrival Time (1/arrival rate)
*       MPT        =  Mean Processing Time
*       NEWAT      =  New Arrival Time
*       OLDAT      =  Old Arrival Time
*       COUNT      =  Counter for zero interarrival times
*       Time Unit  =  Milliseconds
***********************************************************************
        SIMULATE
*
*       Initializations
*
        INITIAL     XH$MIAT,5/XH$MPT,4
*
*       Function Definition
*
 XPDIS FUNCTION    RN1,C24   Exponential Probability Distribution Function
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
*
*       Model Segment
*
 1      GENERATE    XH$MIAT,FN$XPDIS,,,,1PF    Job enters computer system
 2      ASSIGN      1,XH$MPT,XPDIS,PF          Record job processing time
 3      SAVEVALUE   NEWAT,AC1,XF               Record new arrival time (AT)
 4  TSTB TEST E     XF$NEWAT,XF$OLDAT,OKAY     New AT = Old AT ?
 5      SAVEVALUE   COUNT+,1,XF                Count zero interarrival time
 6  OKAY SAVEVALUE  OLDAT,XF$NEWAT,XF          Set old AT = new AT
*
 7      QUEUE       SYSTEM                     Collect statistics
 8      SEIZE       CPU                        Job captures the CPU
 9      ADVANCE     PF1                        CPU processes the job
10      RELEASE     CPU                        Job frees the CPU
11      DEPART      SYSTEM                     Record statistics
12      TERMINATE   1                          Job leaves computer system
*
*       Control Statements
*
        START       50000                      Run for 50,000 jobs
        END                                    End simulation
```

**Figure 4.** An Instrumented GPSS/H Programmed Model

Instead, we should compare the 1,000 with the number of allowable input values and determine how much of the model input domain is covered in testing. The more the model input domain is covered in testing, the more confidence we gain in the accuracy of the model input-output transformation. [Howden 1980; Myers 1979]

### *3.3.2 Bottom-Up Testing*

Bottom-up testing is used in conjunction with bottom-up model development strategy. In bottom-up development, model construction starts with the submodels at the base level (i.e., the ones that are not decomposed further) and culminates with the submodels at the highest level. As each submodel is completed, it is thoroughly tested. When submodels belonging to the same parent have been developed and tested, the submodels are integrated and integration testing is performed. This process is repeated in a bottom-up manner until the whole model has been integrated and tested. The integration of completed submodels need not wait for all “same level” submodels to be completed. Submodel integration and testing can be, and often is, performed incrementally.

Some of the advantages of bottom-up testing are: (1) it encourages extensive testing at the submodel level; (2) since most well-structured models consist of a hierarchy of submodels, there is much to be gained by bottom-up testing; (3) the smaller the submodel and more cohesion it has, the easier and more complete its testing will be; and (4) it is particularly attractive for testing distributed simulation models.

Major disadvantages of bottom-up testing include: (1) individual submodel testing requires drivers, more commonly called test harnesses, which simulate the calling of the submodel and passing test data necessary to execute the submodel; (2) developing harnesses for every submodel can be quite complex and difficult; (3) the harnesses may themselves contain errors; and (4) faces the same cost and complexity problems as does top-down testing.

#### *3.3.3 Debugging*

Debugging is an iterative process the purpose of which is to uncover errors or misconceptions that cause the model’s failure and to define and carry out the model changes that correct the errors. This iterative process consists of four steps. In Step 1, the model is tested revealing the existence of errors (bugs). Given the detected errors, the cause of each error is determined in Step 2. In Step 3, the model changes believed to be required for correcting the detected errors are identified. The identified model changes are carried out in Step 4. Step 1 is re-executed right after Step 4 to ensure successful modification because a change correcting an error may create another one. This iterative process continues until no errors are identified in Step 1 after sufficient testing. [Dunn 1987]

#### *3.3.4 Execution Monitoring*

Execution monitoring is used to reveal errors by examining low-level information about activities and events which take place during model execution. It requires the instrumentation of a programmed or experimental model for the purpose of gathering data to provide activity- or event-oriented information about the model’s dynamic behavior. For example, the instrumented model in Figure 4 monitors the number of zero interarrival times which is important to know for judging the input model validity. The model can also be instrumented to provide other low-level information such as number of jobs with zero processing time, average arrival rate, and average processing time.

### *3.3.5 Execution Profiling*

Execution profiling is used to reveal errors by examining high-level information (profiles) about activities and events which take place during model execution. It requires the instrumentation of a programmed or experimental model for the purpose of gathering data to present profiles about the model's dynamic behavior. For example, the model in Figure 4 can be instrumented to produce the following profiles to assist in model VV&T: (1) a histogram of job interarrival times, (2) a histogram of job processing times, and (3) a histogram of job waiting times in the queue.

### *3.3.6 Execution Tracing*

Execution tracing is used to reveal errors by "watching" the line-by-line execution of a simulation model. It requires the instrumentation of a programmed or experimental model for the purpose of tracing the model's line-by-line dynamic behavior. For example, the model in Figure 4 can be instrumented to record the values of NEWAT in a file during the model execution. Then, the interarrival times can be extracted from the trace data and can be statistically tested to see if they have an exponential probability distribution with mean MIAT *as intended*.

The major disadvantage of the tracing technique is that execution of the instrumented model may produce a large volume of trace data that may be too complex to analyze. To overcome this problem, the trace data can be stored in a database and the modeler can analyze it using a query language [Fairley 1975, 1976].

### *3.3.7 Field Testing*

Field Testing places the model in an operational situation for the purpose of collecting as much information as possible for model validation. It is especially useful for validating models of military combat systems. Although it is usually difficult, expensive, and sometimes impossible to devise meaningful field tests for complex systems, their use wherever possible helps both the project team and decision makers to develop confidence in the model. [Shannon 1975; Van Horn 1971]

### *3.3.8 Graphical Comparisons*

Graphical Comparisons is a subjective, inelegant, and heuristic, yet quite practical approach especially useful as a preliminary approach to model VV&T. The graphs of values of model variables over time are compared with the graphs of values of system variables to investigate characteristics such as similarities in periodicities, skewness, number and location of inflection points, logarithmic rise and linearity, phase shift, trend lines, and exponential growth constants. [Cohen and Cyert 1961; Forrester 1961; Miller 1975; Wright 1972]

### *3.3.9 Predictive Validation*

Predictive Validation requires past data. The model is driven by past system input data and its forecasts are compared with the corresponding past system output data to test the predictive ability of the model. [Emshoff and Sisson 1970]

### *3.3.10 Regression Testing*

Regression testing is used to substantiate that correcting errors and/or making changes in the model do not create other errors and adverse side-effects. It is usually accomplished by retesting the modified model with the previous test data sets used. Successful regression testing requires planning throughout the model development life cycle. Retaining and managing old test data sets are essential for the success of regression testing.

### *3.3.11 Sensitivity Analysis*

Sensitivity Analysis is performed by systematically changing the values of model input variables and parameters over some range of interest and observing the effect upon model behavior [Shannon 1975]. Unexpected effects may reveal invalidity. The input values can also be changed to induce errors to determine the sensitivity of model behavior to such errors. Sensitivity analysis can identify those input variables and parameters to the values of which model behavior is very sensitive. Then, model validity can be enhanced by assuring that those values are specified with sufficient accuracy. [Hermann 1967; Miller 1974a,b; Van Horn 1971]

### *3.3.12 Statistical Techniques*

Much research has been conducted in applying statistical techniques for model validation. Table 1 presents the statistical techniques proposed for model validation and lists related references.

The statistical techniques listed in Table 1 require that the system being modeled is completely observable, i.e., all data required for model validation can be collected from the system. Model validation is conducted by using the statistical techniques to compare the model output data with the corresponding system output data when the model is run with the “same” input data that derive the real system. Due to the multiple response problem [Shannon 1975], the comparison of model and system outputs must be carried out by using a multivariate statistical technique to incorporate the correlations among the output variables.

A validation procedure based on the use of simultaneous confidence intervals is presented below.

| **Table 1.** Statistical Techniques Proposed for Validation | |
|---|---|
| Analysis of Variance | [Naylor and Finger 1967] |
| Confidence Intervals/Regions | [Balci and Sargent 1984; Law and Kelton 1991; Shannon 1975] |
| Factor Analysis | [Cohen and Cyert 1961] |
| Hotelling's $T^2$ Tests | [Balci and Sargent 1981, 1982a, 1982b, 1983; Shannon 1975] |
| Multivariate Analysis of Variance | [Garratt 1974] |
| — Standard MANOVA | |
| — Permutation Methods | |
| — Nonparametric Ranking Methods | |
| Nonparametric Goodness-of-fit Tests | [Gafarian and Walsh 1969; Naylor and Finger 1967] |
| — Kolmogorov-Smirnov Test | |
| — Cramer-Von Mises Test | |
| — Chi-square Test | |
| Nonparametric Tests of Means | [Shannon 1975] |
| — Mann-Whitney-Wilcoxon Test | |
| — Analysis of Paired Observations | |
| Regression Analysis | [Aigner 1972; Cohen and Cyert 1961; Howrey and Kelejian 1969] |
| Theil's Inequality Coefficient | [Kheir and Holmes 1978; Rowland and Holmes 1978; Theil 1961] |
| Time Series Analysis | |
| — Spectral Analysis | [Fishman and Kiviat 1967; Gallant et al. 1974; Howrey and Kelejian 1969; Hunt 1970; Van Horn 1971; Watts 1969] |
| — Correlation Analysis | [Watts 1969] |
| — Error Analysis | [Damborg and Fuller 1976; Tytula 1978] |
| t-Test | [Shannon 1975; Teorey 1975] |

3.3.12.1 A Validation Procedure Using Simultaneous Confidence Intervals

The behavioral accuracy (validity) of a simulation model with multiple outputs can be expressed in terms of the differences between the corresponding model and system output variables when the model is run with the "same" input data and operational conditions that drive the real system. The range of accuracy of the jth model output variable can be represented by the jth confidence interval (c.i.) for the differences between the means of the jth model and system output variables. The simultaneous confidence intervals (s.c.i.) formed by these c.i.'s are called the model range of accuracy (m.r.a.) [Balci and Sargent 1984].

Assume that there are k output variables from the model and k output variables from the system as shown in Figure 1. Let $(\underline{\mu}^m)' = [\mu_1^m, \mu_2^m, \ldots, \mu_k^m]$ and $(\underline{\mu}^s)' = [\mu_1^s, \mu_2^s, \ldots, \mu_k^s]$ be the k dimensional vectors of the population means of the model and system output variables, respectively. Basically, there are three approaches for constructing the s.c.i. to express the m.r.a. for the mean behavior.

In approach I, the m.r.a. is determined by the 100(1 – γ)% s.c.i. for $\underline{\mu}^m - \underline{\mu}^s$ as

$$[\underline{\delta}, \underline{\tau}] \tag{1}$$

where $\underline{\delta}' = [\delta_1, \delta_2, \ldots, \delta_k]$ representing lower bounds and $\underline{\tau}' = [\tau_1, \tau_2, \ldots, \tau_k]$ representing upper bounds of the s.c.i. We can be 100(1 – γ)% confident that the true differences between the population means of the model and system output variables are simultaneously contained within (1).

In approach II, the $100(1 - \gamma^m)$% s.c.i. are first constructed for $\underline{\mu}^m$ as

$$[\underline{\delta}^m, \underline{\tau}^m] \tag{2}$$

where $(\underline{\delta}^m)' = [\delta_1^m, \delta_2^m, \ldots, \delta_k^m]$ and $(\underline{\tau}^m)' = [\tau_1^m, \tau_2^m, \ldots, \tau_k^m]$. Then, the $100(1 - \gamma^s)$% s.c.i. are constructed for $\underline{\mu}^s$ as

$$[\underline{\delta}^s, \underline{\tau}^s] \tag{3}$$

where $(\underline{\delta}^s)' = [\delta_1^s, \delta_2^s, \ldots, \delta_k^s]$ and $(\underline{\tau}^s)' = [\tau_1^s, \tau_2^s, \ldots, \tau_k^s]$. Finally, using the Bonferroni inequality, the m.r.a. is determined by the following s.c.i. for $\underline{\mu}^m - \underline{\mu}^s$ with a confidence level of at least $(1 - \gamma^m - \gamma^s)$ when the model and system outputs are dependent and with a level of at least $(1 - \gamma^m - \gamma^s + \gamma^m\gamma^s)$ when the outputs are independent [Kleijnen 1975]:

$$[\underline{\delta}^m - \underline{\tau}^s, \underline{\tau}^m - \underline{\delta}^s]. \tag{4}$$

In approach III, the model and system output variables are observed in pairs and the m.r.a. is determined by the 100(1 – γ)% s.c.i. for $\underline{\mu}^d$, the population means of the differences of paired observations, as

$$[\underline{\delta}^d, \underline{\tau}^d] \tag{5}$$

where $(\underline{\delta}^d)' = [\delta_1^d, \delta_2^d, \ldots, \delta_k^d]$ and $(\underline{\tau}^d)' = [\tau_1^d, \tau_2^d, \ldots, \tau_k^d]$.

The approach for constructing the m.r.a. should be chosen with respect to the way the model is driven. The m.r.a. is constructed by using the observations collected from the model and system output variables by running the model with the "same" input data and operational conditions that drive the real system. If the simulation model is self-driven, then the "same" indicates that the model input data are coming independently from the same populations or stochastic process of the system input data. Since the model and system input data are independent of each other, but coming from the same populations, the model and system output data are expected to be independent and identically distributed. Hence, approach I or II can be used. The use of approach III in this case would be less efficient. If the simulation model is trace-driven, the "same" indicates that the model input data are exactly the same as the system input data. In this case, the model and system output data are expected to be dependent and identical. Therefore, approach II or III should be used.

Sometimes, the model sponsor, model user, or a third party may specify an acceptable range of accuracy for a specific simulation study. This specification can be made for the mean behavior of a stochastic simulation model as

$$\underline{L} \leq \underline{\mu}^{m} - \underline{\mu}^{s} \leq \underline{U} \qquad (6)$$

where $\underline{L}' = [L_1, L_2, \ldots, L_k]$ and $\underline{U}' = [U_1, U_2, \ldots, U_k]$ are the lower and upper bounds of the acceptable differences between the population means of the model and system output variables. In this case, the m.r.a. should be compared against (6) to evaluate model validity.

The shorter the lengths of the m.r.a., the more meaningful is the information they provide. The lengths can be decreased by increasing the sample sizes or by decreasing the confidence level. However, such increases in sample sizes may increase the cost of data collection. Thus, a trade-off analysis may be necessary among the sample sizes, confidence levels, half length estimates of the m.r.a, data collection method, and cost of data collection. For details of performing the trade-off analysis see [Balci and Sargent 1984].

The confidence interval validation procedure is presented in Figure 5.

### *3.3.13 Stress Testing*

Stress testing is intended to test the model validity under extreme workload conditions. This is usually accomplished by increasing the congestion in the model. For example, the model in Figure 4 can be stress tested by decreasing the mean interarrival time (MIAT), i.e., bringing in more jobs per unit time. Such increase in workload may create high congestion and result in unacceptably high number of zero interarrival times causing incorrect representation of the input Poisson arrival process. Under stress testing, the model may exhibit invalid behavior; however, such behavior should be as expected and meaningfully documented. Without the instrumentation, the model in Figure 4 could produce inaccurate results under such stress testing. [Dunn 1987; Myers 1979]

### *3.3.14 Submodel Testing*

Submodel Testing requires a top-down model decomposition in terms of submodels. The experimental model is instrumented to collect data on all input and output variables of a submodel. The system is similarly instrumented (if possible) to collect similar data. Then, each submodel behavior is compared with corresponding subsystem behavior to judge submodel validity. If a subsystem can be modeled analytically (e.g., as an M/M/1 model), its exact solution can be compared against the simulation solution to assess validity quantitatively.

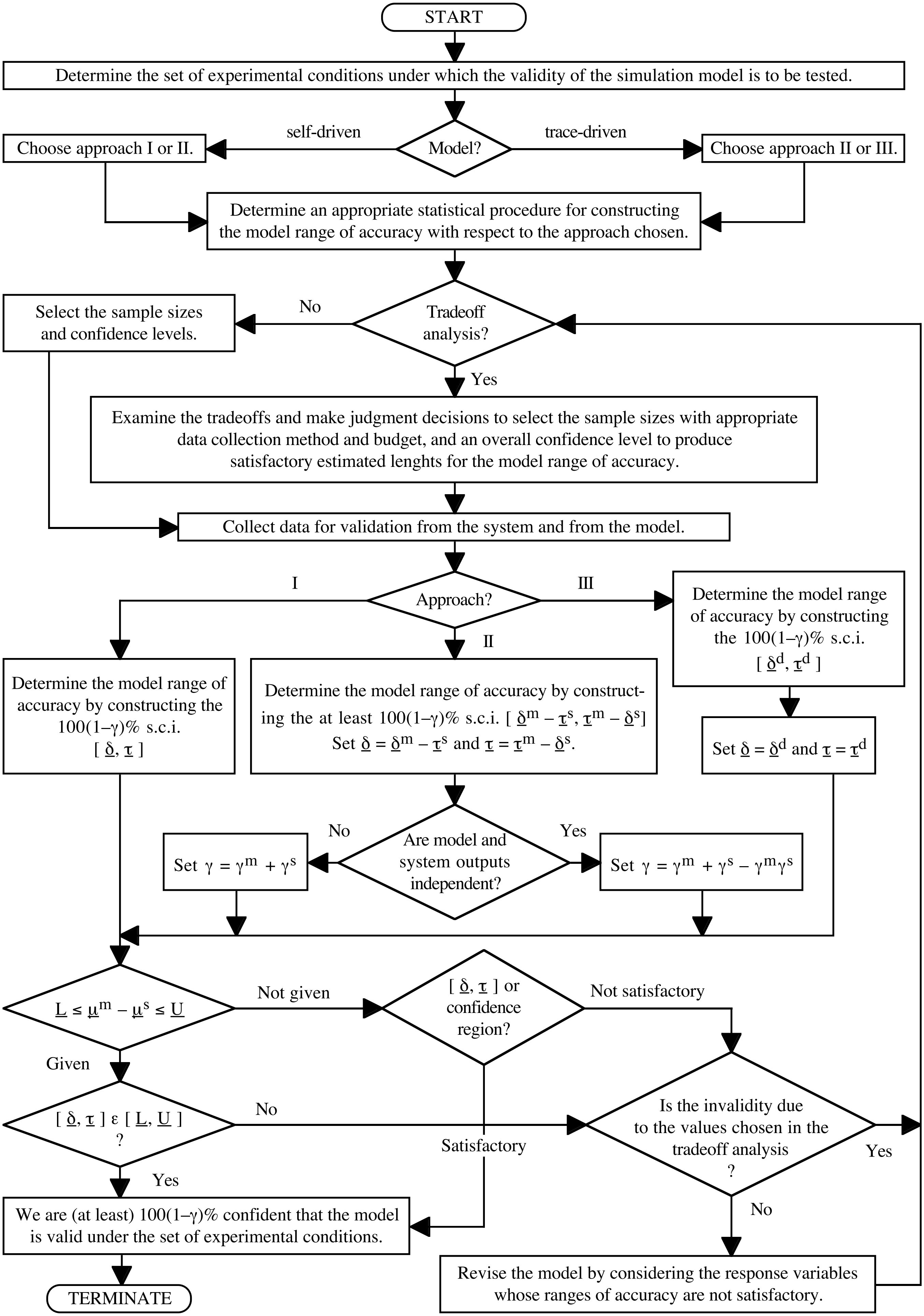


**Figure 5.** A Validation Procedure Using Simultaneous Confidence Intervals

Validating each submodel individually does not imply sufficient validity for the whole model; because, each submodel validity is passed with some error tolerance and the allowable errors can accumulate to make the whole model invalid. Therefore, after individually validating each submodel, the whole model itself must be subjected to testing.

### *3.3.15 Symbolic Debugging*

Symbolic debugging assists in model VV&T by employing a debugging tool that allows the modeler to manipulate model execution while viewing the model at the source code level. By setting "breakpoints", the modeler can interact with the entire model one step at a time, at predetermined locations, or under specified conditions. While using a symbolic debugger, the modeler may alter model data values or cause a portion of the model to be "replayed", i.e., executed again under the same conditions (if possible). Typically, the modeler utilizes the information from execution history generation techniques, such as tracing, monitoring, and profiling, to isolate a problem or its proximity. He then proceeds with the debugger to understand how and why the error occurs.

Current state-of-the-art debuggers allow viewing the runtime code as it appears in the source listing, setting "watch" variables to monitor data flow, viewing complex data structures, and even communicating with asynchronous I/O channels. The use of symbolic debugging can greatly reduce the debugging effort while increasing its effectiveness. Symbolic debugging allows the modeler to locate errors and check numerous circumstances which lead up to the errors. [Whitner and Balci 1989]

### *3.3.16 Top-Down Testing*

Top-down testing is used in conjunction with top-down model development strategy. In top-down development, model construction starts with the submodels at the highest level and culminates with the submodels at the base level (i.e., the ones that are not decomposed further). As each submodel is completed, it is thoroughly tested. When submodels belonging to the same parent have been developed and tested, the submodels are integrated and integration testing is performed. This process is repeated in a top-down manner until the whole model has been integrated and tested. The integration of completed submodels need not wait for all "same level" submodels to be completed. Submodel integration and testing can be, and often is, performed incrementally.

Top-down testing begins with testing the global model at the highest level. When testing a given level, calls to submodels at lower levels are simulated using submodel "stubs." A stub is a dummy submodel which has no other function than to let its caller complete the call. Fairley [1976] lists the following advantages of top-down testing: (1) model integration testing is minimized, (2) early existence of a working model results, (3) higher level interfaces are tested first, (4) a natural environment for testing lower levels is provided, and (5) errors are localized to new submodels and interfaces.

Some of the disadvantages of top-down testing are: (1) thorough submodel testing is discouraged (the entire model must be executed to perform testing), (2) testing can be expensive (since the whole model must be executed for each test), (3) adequate input data is difficult to obtain (because of the complexity of the data paths and control predicates), and (4) integration testing is hampered (again, because of the size and complexity induced by testing the whole model) [Fairley 1976].

#### *3.3.17 Visualization*

Visualization (animation) of a simulation model greatly assists in model VV&T [Sargent 1992]. Displaying graphical images of internal and external dynamic behavior of a model during execution enables us to discover errors by seeing. For example, in visual simulation of a traffic intersection, we can observe the arrivals of vehicles in different lanes and their movements through the intersection as the traffic light changes. Seeing the visualization of the model as it executes and comparing it with the operations of the real traffic intersection can help us identify discrepancies between the model and the system.

Seeing the model in action is very useful for uncovering errors; however, seeing is not believing here [Paul 1989]. Seeing model behavior during execution does not guarantee model correctness. Therefore, visualization should be used with caution in model VV&T.

#### *3.3.18 White-Box Testing*

White-box testing is used to evaluate the model based on its internal structure (how it is built) whereas black-box testing is intended for assessing the input-output transformation accuracy of the model. White-box testing employs data flow and control flow diagrams to assess the accuracy of internal model structure by examining model elements such as internal logic, internal data representations, submodel interfaces, and model execution paths. White-box testing is quite effective for detecting redundant code, faulty model structure, and special case errors.

### 3.4 Symbolic VV&T Techniques

Symbolic VV&T techniques, like dynamic VV&T techniques, are used to evaluate the dynamic behavior of the model during execution.

#### *3.4.1 Cause-Effect Graphing*

Cause-effect graphing assists model VV&T by addressing the question of “what causes what in the model representation?” It is performed by first identifying causes and effects in the system being modeled and by examining if they are accurately reflected in the model specification. For example, in the simulation of a traffic intersection, the following causes and effects may be identified: (1) the change of lane 1 light to red immediately causes the vehicles in lane 1 to stop, (2) an increase in the duration of lane 1 green light causes a decrease in the average waiting time of vehicles in lane 1, and (3) an increase in the arrival rate of lane 1 vehicles causes an increase in the average number of vehicles at the intersection.

As many causes and effects as possible are listed, and the semantics are expressed in a cause-effect graph. The graph is annotated to describe special conditions or impossible situations. Once the cause-effect graph has been constructed, a decision table is created by tracing back through the graph to determine combinations of causes which result in each effect. The decision table is then converted into test cases with which the model is tested. [Myers 1979; Whitner and Balci 1989]

### *3.4.2 Partition Analysis*

Partition analysis is used for testing the model with the test data generated by analyzing the model's functional representatives (partitions). It is accomplished by: (1) decomposing both model specification and implementation into functional representatives (partitions), (2) comparing the elements and prescribed functionality of each partition specification with the elements and actual functionality of corresponding partition implementation, (3) deriving test data to extensively test the functional behavior of each partition, and (4) testing the model by using the generated test data.

The model decomposition into functional representatives (partitions) is derived through the use of symbolic evaluation techniques which maintain algebraic expressions of model elements and show model execution paths. These functional representations are the model computations. Two computations are equivalent if they are defined for the same subset of the input domain which causes a set of model paths to be executed, and if the result of the computations is the same for each element within the subset of the input domain [Howden 1976]. Standard proof techniques are used to show equivalence over a domain. When equivalence cannot be shown, partition testing is performed to locate errors, or as Richardson and Clarke [1985] state, to increase confidence in the equality of the computations due to the lack of error manifestation. By involving both model specification and implementation, partition analysis is capable of providing more comprehensive test data coverage than other test data generation techniques.

### *3.4.3 Path Analysis*

Path analysis [Howden 1976] attempts to assess model correctness on the basis of complete testing of all model control paths. It is performed in three steps. In Step 1, the model control structure (e.g., through structural analysis) is determined and represented in a control flow diagram. In Step 2, test data is generated to cause selected model logical paths to be executed. Symbolic execution can be used to identify and group together classes of input data based on the symbolic representation of the model. The test data is generated in such a way as to: (1) cover all statements in the path, (2) encounter all nodes in the path, (3) cover all branches from a node in the path, (4) achieve all decision combinations at each branch point in the path, and (5) traverse all paths [Prather and Myers 1987]. In Step 3, by using the generated test data, the model is forced to proceed through each path in its execution structure, thereby providing comprehensive testing.

In practice, only a subset of all possible model paths are selected for testing due to the budgetary constraints. Recent work has sought to increase the amount of coverage per test case or to improve the effectiveness of the testing by selecting the most critical areas to test. The path prefix strategy is an “adaptive” strategy that uses previous paths tested as a guide in the selection of subsequent test paths. Prather and Myers [1987] prove that the path prefix strategy achieves total branch coverage.

The identification of essential paths is a strategy which reduces the path coverage required by nearly 40 percent [Chusho 1987]. The basis for the reduction is the elimination of non-essential paths. Paths which are overlapped by other paths are non-essential. The model control flow graph is transformed into a directed graph whose arcs (called primitive arcs) correspond to the essential paths of the model. Non-essential arcs are called inheritor arcs because they inherit information from the primitive arcs. The graph produced during the transformation is called an inheritor-reduced graph. Chusho [1987] presents algorithms for efficiently identifying non-essential paths and reducing the control graph into an inheritor-reduced graph, and for applying the concept of essential paths to the selection of effective test data.

### *3.4.4 Symbolic Execution*

Symbolic execution is used to assess model accuracy by executing the model using symbolic values rather than actual data values for input. It is performed by feeding symbolic inputs into the (sub)model and producing expressions for the output which are derived from the transformation of the symbolic data along model execution paths. Consider, for example, the following function:

```
function jobArrivalTime(arrivalRate,currentClock,randomNumber)
    lag = -10
    Y = lag * currentClock
    Z = 3 * Y
    if Z < 0 then
        arrivalTime = currentClock - log(randomNumber) / arrivalRate
    else
        arrivalTime = Z - log(randomNumber) / arrivalRate
    end if
    return arrivalTime
end jobArrivalTime
```

In symbolic execution, `lag` is substituted in `Y` resulting in `Y = -10*currentClock`. Substituting again, we find `Z = -30*currentClock`. Since `currentClock` is always zero or positive, an error is detected that `Z` will never be greater than zero.

When unresolved conditional branches are encountered, a decision must be made which path to traverse. Once a path is selected, execution continues down the new path. At some point in time, the execution evaluation will return to the branch point and the previously unselected branch will be traversed. All paths eventually are taken.

The result of the execution can be represented graphically as a symbolic execution tree [King 1976; Adrion et al. 1982]. The branches of the tree correspond to the paths of the model. Each node of the tree represents a decision point in the model and is labeled with the symbolic values of data at that juncture. The leaves of the tree are complete paths through the model and depict the symbolic output produced.

Symbolic execution assists in showing path correctness for all computations regardless of test data and is also a great source of documentation. However, it has the following disadvantages: (1) the execution tree can explode in size and become too complex as the model grows, (2) loops cause difficulties although inductive reasoning and constraint analysis may help, (3) loops make thorough execution impossible since all paths must be traversed, and (4) complex data structures may have to be excluded because of difficulties in symbolically representing particular data elements within the structure. [Dillon 1990; King 1976; Ramamoorthy et al. 1976]

## 3.5 Constraint VV&T Techniques

Constraint VV&T techniques are employed to assess model correctness using assertion checking, boundary analysis, and inductive assertions.

### *3.5.1 Assertion Checking*

An *assertion* is a statement that should hold true as the simulation model executes. Assertion checking is a verification technique used to check what *is* happening against what the modeler *assumes* is happening so as to guard model execution against potential errors. The assertions are placed in various parts of the model to monitor model execution. They can be inserted to hold true *globally*—for the whole model; *regionally*—for some submodels; *locally*—within a submodel; or at *entry* and *exit* of a submodel. The assertions are similar in structure and the general format for a local assertion is [Stucki 1977]:

```
ASSERT LOCAL (extended-logical-expression) [optional-qualifiers]
                                            [control-options]
```

The "optional-qualifiers" may be chosen as `all`, `some`, `after jth job`, `before time t`, etc. The "control-options" may have the following example syntax [Stucki 1977]:

$$\cdots \texttt{ LIMIT n [VIOLATIONS]} \left[\left\{\begin{array}{l}\underline{\texttt{HALT}}\\ \texttt{EXIT [VIA] proc-name}\end{array}\right\}\right] \cdots$$

For example, the programmed model in Figure 4 can be instrumented by inserting the following local assertion right after the 5th Block to stop the simulation if the frequency of violation exceeds an allowable value of 1%:

```
ASSERT LOCAL((100*XF$COUNT/N$TSTB)'LE'1) LIMIT 1
```

where N$TSTB is a GPSS/H Standard Numerical Attribute giving the total number of entries into the Block labeled as TSTB.

Consider, for example, the following pseudo-code [Whitner and Balci 1989]:

```
Base := Hours * PayRate;
Gross := Base * (1 + BonusRate);
```

In just these two simple statements, several assumptions are being made. It is assumed that `Hours`, `PayRate`, `Base`, `BonusRate`, and `Gross` are all non-negative. The following asserted code can be used to prevent execution errors due to incorrect values inputted by the user:

```
Assert Local (Hours ≥ 0 and PayRate ≥ 0 and BonusRate ≥ 0);
Base := Hours * PayRate;
Gross := Base * (1 + BonusRate);
```

Clearly, the assertion checking serves two important needs: (1) it verifies that the model is functioning within its acceptable domain, and (2) the assertion statement documents the intentions of the modeler. However, the assertion checking degrades the model execution performance. If the execution performance is critical, the assertions should be labeled as comments and should be kept permanently to provide both documentation and means for maintenance testing [Adrion et al. 1982].

Usually, a language should be defined for assertion specification [Howden 1980] and a preprocessor is needed to translate assertions into the simulation programming language. Unfortunately, most of the current simulation programming languages do not provide assertion checking although this notion is not at all new dating back to 1972 when Satterthwaite included an ASSERT statement in his version of Algol W [Satterthwaite 1972].

### *3.5.2 Boundary Analysis*

Boundary analysis is employed to test model accuracy by using test cases on the boundaries of input equivalence classes. A model's input domain can usually be divided into classes of input data (known as equivalence classes) which cause the model to function the same way. For example, a traffic intersection model might specify the probability of left turn in a three-way turning lane as 0.2, the probability of right turn as 0.35, and the probability of travelling straight as 0.45. This probabilistic branching can be implemented by using a uniform random number generator that produces numbers in the range $0 \le rn \le 1$. Thus, three equivalence classes are identified: $0 \le rn \le 0.2$, $0.2 < rn \le 0.55$, and $0.55 < rn \le 1$. Each test case from within a given equivalence class has the same effect on the model behavior, i.e., produces the same direction of turn.

In boundary analysis, test cases are generated just within, on top of, and just outside of the equivalence classes [Myers 1979]. In the example above, the following test cases are selected for the left turn: 0.0, ± 0.000001, 0.199999, 0.2, and 0.200001. In addition to generating test data on the basis of input equivalence classes, it is also useful to generate test data which will cause the model to produce values on the boundaries of output equivalence classes [Myers 1979]. The underlying rationale for this technique as a whole is that the most error-prone test cases lie along the boundaries [Ould and Unwin 1986]. Notice that invalid test cases used in the example above will cause the model execution to fail; however, this failure should be as expected and meaningfully documented.

#### *3.5.3 Inductive Assertions*

Inductive assertions are used to assess model correctness based on an approach that is very close to formal proof of model correctness. It is conducted in three steps. In Step 1, input-to-output relations for all model variables are identified. In Step 2, these relations are converted into assertion statements and are placed along the model execution paths in such a way as to divide the model into a finite number of "assertion-bound" paths, i.e., an assertion statement lies at the beginning and end of each model execution path. In Step 3, verification is achieved by proving that for each path: if the assertion at the beginning of the path is true, and all statements along the path are executed, then the assertion at the end of the path is true. If all paths plus model termination can be proved, by induction, the model is proved to be correct. [Manna et al. 1973; Reynolds and Yeh 1976]

### 3.6 Formal VV&T Techniques

Formal VV&T techniques are based on formal mathematical proof of correctness. If attainable, formal proof of correctness is the most effective means of model VV&T. Unfortunately, "if attainable" is the overriding point with regard to formal VV&T techniques. Current state-of-the-art formal proof of correctness techniques are simply not capable of being applied to even a reasonably complex simulation model. However, formal techniques serve as the foundation for other VV&T techniques and the most commonly known seven techniques are covered below: (1) induction, (2) inference, (3) λ-calculus, (4) logical deduction, (5) predicate calculus, (6) predicate transformation, and (7) proof of correctness. [Khanna 1991; Whitner and Balci 1989]

*Induction*, *inference*, and *logical deduction* are simply acts of justifying conclusions on the basis of premises given. An argument is valid if the steps used to progress from the premises to the conclusion conform to established *rules of inference*. Inductive reasoning is based on invariant properties of a set of observations (assertions are invariants since their value is defined to be true). A typical inductive argument would be one similar to the one given in Section 3.5.3 for inductive assertions: given that the initial model assertion is correct, it stands to reason that if each path progressing from that assertion can be shown to be correct, and subsequently each path progressing from the previous assertion is correct, etc., then the model must be correct if it terminates. Formal induction proof techniques exist for the intuitive explanation just given.

The *λ-calculus* [Barendregt 1981] is a system for transforming the model into formal expressions. It is a string-rewriting system and the model itself can be considered as a large string. The λ-calculus specifies rules for rewriting strings, i.e., transforming the model into λ-calculus expressions. Using the λ-calculus, the modeler can formally express the model so that mathematical proof of correctness techniques can be applied.

The *predicate calculus* provides rules for manipulating predicates. A predicate is a combination of simple relations, such as *completed_jobs >steady_state_length*. A predicate will either be true or false. The model can be defined in terms of predicates and manipulated using the rules of the predicate calculus. The predicate calculus forms the basis of all formal specification languages [Backhouse 1986]. *Predicate transformation* [Dijkstra 1975; Yeh 1977] provides a basis for verifying model correctness by formally defining the semantics of the model with a mapping which transforms model output states to all possible model input states. This representation provides the basis for proving model correctness.

Formal *proof of correctness* corresponds to expressing the model in a precise notation and then mathematically proving that the executed model terminates and it satisfies the requirements specification with sufficient accuracy [Backhouse 1986]. Attaining proof of correctness in a realistic sense is not possible under the current state of the art. However, the advantage of realizing proof of correctness is so great that when the capability is realized, it will revolutionize the model VV&T.

## 4. CREDIBILITY ASSESSMENT STAGES

It is very important to understand the principles of simulation model VV&T when applying the VV&T techniques throughout the entire life cycle of a simulation study. Balci [1994] presents 15 principles that help the researchers, practitioners and managers better understand what model VV&T is all about. These principles serve to provide the underpinnings for the VV&T techniques described in Section 3. Understanding and applying these principles is crucially important for the success of a simulation study.

Table 2 marks the VV&T techniques that are applicable for each of the ten credibility assessment stages described below. The more of these techniques we apply the more confidence we gain in the credibility of a life cycle phase. The VV&T activities should continue until a sufficient level of confidence is achieved.

**Table 2.** Applicability of the VV&T Techniques for the Credibility Assessment Stages

| | FP VV&T | FA of Sim. | S&OD VV&T | Model Qual. | CM VV&T | PM VV&T | ED VV&T | Data VV&T | EM VV&T | Pres. VV&T |
|---|---|---|---|---|---|---|---|---|---|---|
| Assertion Checking | | | | | | ✔ | ✔ | ✔ | ✔ | |
| Audit | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Black-Box Testing | | | | | | ✔ | ✔ | | ✔ | |
| Bottom-Up Testing | | | | | | ✔ | | | ✔ | |
| Boundary Analysis | | | | | | ✔ | | | ✔ | |
| Cause-Effect Graphing | ✔ | | | | ✔ | ✔ | | | ✔ | |
| Consistency Checking | ✔ | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Data Flow Analysis | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | |
| Debugging | | | | | | ✔ | ✔ | | ✔ | |
| Desk Checking | ✔ | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Execution Monitoring | | | | | | ✔ | ✔ | | ✔ | |
| Execution Profiling | | | | | | ✔ | ✔ | | ✔ | |
| Execution Tracing | | | | | | ✔ | ✔ | | ✔ | |
| Face Validation | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Field Testing | | | | | | | | | ✔ | |
| Graph-Based Analysis | | | | | ✔ | ✔ | ✔ | | ✔ | |
| Graphical Comparisons | | | | | | ✔ | ✔ | | ✔ | |
| Induction | | | | | | ✔ | | | ✔ | |
| Inductive Assertions | | | | | | ✔ | | | ✔ | |
| Inference | | | | | | ✔ | | | ✔ | |
| Inspections | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Lamda Calculus | | | | | | ✔ | | | ✔ | |
| Logical Deduction | | | | | | ✔ | | | ✔ | |
| Partition Analysis | | | | | | ✔ | | | ✔ | |
| Path Analysis | | | | | | ✔ | ✔ | | ✔ | |
| Predicate Calculus | | | | | | ✔ | | | ✔ | |
| Predicate Transformation | | | | | | ✔ | | | ✔ | |
| Predictive Validation | | | | | | | | | ✔ | |
| Proof of Correctness | | | | | | ✔ | | | ✔ | |
| Regression Testing | | | | | | ✔ | | | ✔ | |
| Reviews | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Semantic Analysis | | | | | | ✔ | ✔ | | ✔ | |
| Sensitivity Analysis | | | | | | ✔ | ✔ | | ✔ | |
| Statistical Techniques | | | | | | | | ✔ | ✔ | |
| Stress Testing | | | | | | ✔ | | | ✔ | |
| Structural Analysis | ✔ | | ✔ | | ✔ | ✔ | ✔ | | ✔ | ✔ |
| Submodel Testing | | | | | | ✔ | | | ✔ | |
| Symbolic Debugging | | | | | | ✔ | ✔ | | ✔ | |
| Symbolic Execution | | | | | | ✔ | ✔ | | ✔ | |
| Syntax Analysis | | | | | | ✔ | ✔ | | ✔ | |
| Top-Down Testing | | | | | | ✔ | | | ✔ | |
| Turing Test | | | | | | | | | ✔ | |
| Visualization | | | | | | ✔ | | | ✔ | |
| Walkthroughs | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| White-Box Testing | | | | | | ✔ | | | ✔ | |

## 4.1 Formulated Problem VV&T

When a problem is recognized, a decision maker (a client or sponsor group) initiates a study by communicating the problem to an analyst (a problem-solver or a consultant/research group). The communicated problem is rarely clear, specific, or organized. Consequently, an essential study to formulate the *actual* problem usually follows. *Problem Formulation* (problem structuring or problem definition) is the process by which the initially communicated problem is translated into a formulated problem sufficiently well defined to enable specific research action [Woolley and Pidd 1981].

Formulated problem VV&T deals with substantiating that the formulated problem contains the *actual* problem in its entirety and is sufficiently well structured to permit the derivation of a sufficiently credible solution [Balci and Nance 1985]. Failure to formulate the *actual* problem results in the Type III error. Once the Type III error is committed, regardless of how well the problem is solved, the simulation study will either end unsuccessfully or with the Type II error. Therefore, the accuracy of formulated problem greatly affects the credibility and acceptability of simulation results.

Audit, cause-effect graphing, consistency checking, desk checking, face validation, inspections, reviews, structural analysis, and walkthroughs can be applied for conducting formulated problem VV&T. In applying cause-effect graphing, a causality network is created to analyze the potential root causes of the communicated problem [Balci and Nance 1985]. The questionnaire developed by Balci and Nance [1985] with 38 indicators can be used in applying audit, inspections, reviews, and walkthroughs.

## 4.2 Feasibility Assessment of Simulation

All alternative techniques that can be used in solving the formulated problem should be identified. A technique whose solution is estimated to be too costly or is judged to be not sufficiently beneficial with respect to the study objectives should be disregarded. Among the qualified ones, the technique with the highest expected benefits/cost ratio should be selected.

The statement "when all else fails, use simulation" is misleading if not invalid. The question is not to bring a solution to the problem, but to bring a sufficiently credible one which will be accepted and used by the decision maker(s). A technique other than simulation may provide a less costly solution, but it may not be as useful.

Sometimes, the communicated problem is formulated under the influence of a solution technique in mind. Occasionally, simulation is chosen without considering any other technique just because it is the only one the analyst(s) can use. Skipping the investigation of solution techniques process may result in unnecessarily expensive solutions, sometimes to the wrong problems.

As a result of the investigation of solution techniques process, we assume that simulation is chosen as the most appropriate solution technique. At this point, the simulation project team should be activated and be made responsible for the formulated problem VV&T and feasibility assessment of simulation before proceeding in the life cycle.

Audit, face validation, inspections, reviews, and walkthroughs can be applied for assessing the feasibility of simulation with the use of indicators such as: (1) Are the benefits and cost of simulation solution estimated correctly?; (2) Do the potential benefits of simulation solution justify the estimated cost of obtaining it?; (3) Is it possible to solve the problem using simulation within the time limit specified?; (4) Can all of the resources required by the simulation project be secured?; and (5) Can all of the specific requirements (e.g., access to pertinent classified information) of the simulation project be satisfied?

### 4.3 System and Objectives Definition VV&T

For the purpose of generality, the term "system" is used to refer to the entity that contains the formulated problem. System and objectives definition VV&T deals with assessing the credibility of the system investigation process in which system characteristics are explored for consideration in system definition and modeling. Shannon [1975] identifies six major system characteristics: (1) change, (2) environment, (3) counterintuitive behavior, (4) drift to low performance, (5) interdependency, and (6) organization. Each characteristic should be examined with respect to the study objectives that are identified with the formulation of the problem.

In simulation, we mostly deal with stochastic and dynamic real systems that *change* over a period of time. How often and how much the system will change during the course of a simulation study should be estimated so that the model representation can be updated accordingly. Changes in the system may also change the study objectives.

A system's *environment* consists of all input variables that can significantly affect its state. The input variables are identified by assessing the significance of their influence on the system's state with regard to the study objectives. For example, for a traffic intersection system, the interarrival time of vehicles can be identified as an input variable making up the environment, whereas pedestrian arrivals may be omitted due to their negligible effect on the system's behavior of interest (e.g., average waiting time of vehicles). Underestimating the influence of an input variable may result in inaccurate environment definition.

Some complex systems may show *counterintuitive behavior* which we should try to identify for consideration in defining the system. However, this is not an easy task, especially for those systems containing many subjective elements (e.g., social systems). Cause and effect are often not closely related in time or space. Symptoms may appear long after the primary causes [Shannon 1975]. To be able to identify counterintuitive behavior, it is essential that the simulation project employs people who have expert knowledge about the system under study.

A system may show a *drift to low performance* due to the deterioration of its components (e.g., machines in a manufacturing system) over a period of time. If this characteristic exists, it should be incorporated within the model representation especially if the model's intended use is forecasting.

Before we start abstracting the real system for the purpose of modeling, we should examine the *interdependency* and *organization* characteristics of the system. In a complex stochastic system, many activities or events take place simultaneously and influence each other. The system complexity can be overcome by way of decomposing the system into subsystems and subsystems into other subsystems. This decomposition can be carried out by examining how system elements or components are organized.

Audit, consistency checking, desk checking, face validation, inspections, reviews, structural analysis, and walkthroughs can be applied for conducting system and objectives definition VV&T by using indicators such as: (1) Since systems and objectives may change over a period of time, will we have the same system and objectives definition at the conclusion of the simulation study (which may last from 6 months to several years)?; (2) Is the system's environment (boundary) identified correctly?; (3) What counterintuitive behavior may be caused within the system and its environment?; (4) Will the system significantly drift to low performance requiring a periodic update of the system definition?; and (5) Are the interdependency and organization of the system characterized accurately? The objective here is to substantiate that the system characteristics are identified and the study objectives are explicitly defined with sufficient accuracy. An error made here may not be caught until very late in the life cycle resulting in a high cost of correction or an error of type II or III.

### 4.4 Model Qualification

Model qualification is intended for assessing the credibility of the model formulation process. Model formulation is the process by which a conceptual model is envisioned to represent the system under study. The *Conceptual Model* is the model which is formulated in the mind of the modeler [Nance 1981]. Model formulation and model representation constitute the process of model design. *Input data analysis and modeling* [Law and Kelton 1991] is a subprocess of model formulation process and is conducted with respect to the way the model is driven, self- or trace-driven.

Under some study objectives (e.g., evaluation, comparison, determination of functional relations) and for model validation, input data model is built to represent the system's input process. In a self-driven simulation (e.g., of a computer system), we collect data on an input random variable (e.g., inter-arrival time of jobs), identify the distribution, estimate its parameters, and conclude upon a probability distribution as the input data model to sample from in driving the simulation model [Law and Kelton 1991]. In a trace-driven simulation, we trace the system (e.g., using hardware or software monitors) and utilize the refined trace data as the input data model to use in driving the simulation model.

A model should be conceptualized under the guidance of a structured approach such as the Conical Methodology [Nance 1994, 1987, 1981]. One key idea behind the use of a structured approach is to control the model complexity so that we can successfully verify and validate the model. The use of a structured approach is an important factor determining the success of a simulation project, especially for large-scale and complex models. During the conceptualization of the model, one makes many assumptions in abstracting the reality. Each assumption should be explicitly specified.

Model Qualification deals with the justification that all assumptions made are appropriate and the conceptual model provides an adequate representation of the system with respect to the study objectives. Audit, consistency checking, desk checking, face validation, inspections, reviews, and walkthroughs can be applied for conducting model qualification.

### 4.5 Communicative Model VV&T

Communicative model VV&T is concerned with substantiating the sufficient accuracy of the model representation process which is the process of translating the conceptual model into a communicative model. A *Communicative Model* is "a model representation which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human" [Nance 1981]. Several communicative models may be developed; one in the form of Structured English intended for nontechnical people, another in the form of a micro flowchart intended for a programmer.

Communicative model VV&T deals with confirming the adequacy of the communicative model to provide an acceptable level of agreement for the domain of intended application. *Domain of Intended Application* [Schlesinger et al. 1979] is the prescribed conditions for which the model is intended to match the system under study. *Level of Agreement* [Schlesinger et al. 1979] is the required correspondence between the model and the system, consistent with the domain of intended application and the study objectives.

Audit, cause-effect graphing, consistency checking, data flow analysis, desk checking, face validation, graph-based analysis, inspections, reviews, structural analysis, and walkthroughs can be applied for conducting communicative model VV&T.

### 4.6 Programmed Model VV&T

Programmed model VV&T deals with the assessment of the process of programming which is the process of translating the communicative model into a programmed model. A *Programmed Model* is an executable simulation model representation in a simulation programming language such as GPSS, SIMAN, SIMSCRIPT, SIMULA, and SLAM or in a high-level programming language such as C++, Fortran, and Pascal. The programmed model does not incorporate an experiment design.

The techniques applicable for conducting communicative model VV&T are marked in Table 2.

### 4.7 Experiment Design VV&T

Experiment design VV&T deals with substantiating the sufficient accuracy of the process of design of experiments which is the process of formulating a plan to gather the desired information at minimal cost and to enable the analyst to draw valid inferences [Shannon 1975]. An *Experimental Model* is the programmed model incorporating an executable description of operations presented in such a plan.

A variety of techniques are available for the design of experiments. *Response-surface methodologies* can be used to find the optimal combination of parameter values which maximize or minimize the value of a response variable [Law and Kelton 1991]. *Factorial designs* can be employed to determine the effect of various input variables on a response variable [Fishman 1978]. *Variance reduction techniques* can be implemented to obtain greater statistical accuracy for the same amount of simulation [Law and Kelton 1991]. *Ranking and selection techniques* can be utilized for comparing alternative systems [Law and Kelton 1991; Banks and Carson 1984]. Several methods (e.g., replication, batch means, regenerative) can be used for statistical analysis of simulation output data.

The techniques marked in Table 2 can be applied for conducting experiment design VV&T with the use of indicators such as: (1) Are the algorithms used for random variate generation theoretically accurate?; (2) Are the random variate generation algorithms translated into executable code accurately? (Error may be induced by computer arithmetic or by truncation due to machine accuracy, especially with order statistics (e.g., $X = -\log_e(1-U)$) [Schmeiser 1981].); (3) How well is the random number generator tested? (Using a generator which is not rigorously shown to produce uniformly distributed independent numbers with sufficiently large period may invalidate the whole experiment design.); (4) Are *appropriate* statistical techniques implemented to design and analyze the simulation experiments? How well are the underlying assumptions satisfied? (See [Law 1983] for several reasons why output data analyses have not been conducted in an appropriate manner.); (5) Is the problem of the initial transient (or the start-up problem) [Wilson and Pritsker 1978] appropriately addressed?; and (6) For comparison studies, are identical experimental conditions replicated correctly for each of the alternative operating policies compared?

### 4.8 Data VV&T

Data VV&T involves input data model VV&T and deals with substantiating that all data used throughout the model development phases of the life cycle in Figure 2 are accurate, complete, unbiased, and appropriate in their original and transformed forms. An input data model is the characterization of an input process (e.g., characterization of an arrival process by Poisson probability distribution). U.S. GAO [1987] emphasizes the importance of input data model validation in credibility assessment of simulations.

In those cases where data cannot be collected, data values may be determined through calibration. *Calibration* is an iterative process in which a probabilistic characterization for an input variable or a fixed value for a parameter is tried until the model is found to be sufficiently valid.

Assertion checking, audit, consistency checking, data flow analysis, desk checking, face validation, inspections, reviews, statistical techniques, and walkthroughs can be applied for conducting data VV&T with the use of indicators such as: (1) Does each input data model possess a sufficiently accurate representation?; (2) Are the parameter values identified, measured, or estimated with sufficient accuracy?; (3) How reliable are the instruments used for data collection and measurement?; (4) Are all data transformations done accurately? (e.g., are all data transformed correctly into the same time unit of the model?); (5) Is the dependence between the input variables, if any, represented by the input data model (s) with sufficient accuracy? (Blindly modeling bivariate relationships using only correlation to measure dependency is cited as a common error by Schmeiser [1981].); and (6) Are all data up-to-date?

### 4.9 Experimental Model VV&T

Experimental Model VV&T deals with substantiating that the experimental model has sufficient accuracy in representing the system as defined under the study objectives. All of the 45 techniques listed in Table 2 can be applied for conducting model VV&T. The applicability of the 45 techniques depends upon the following cases where the system being modeled is: (1) completely observable—all data required for model VV&T can be collected from the system, (2) partially observable—some required data can be collected, or (3) nonexistent or completely unobservable. The statistical techniques in Table 1 are applicable only for case 1.

### 4.10 Presentation VV&T

Presentation VV&T deals with justifying that the simulation results are interpreted, documented, and communicated with sufficient accuracy.

Since all simulation models are descriptive, simulation results must be interpreted. (A descriptive model describes the behavior of a system without any value judgment on the “goodness” or “badness” of such behavior [Elmaghraby 1968].) In the simulation of an interactive computer system, for example, the model may produce a value of 20 seconds for the average response time; but, it does not indicate whether the value 20 is a “good” result or a “bad” one. Such a judgment is made by the simulation analyst depending upon the study objectives. Under one set of study objectives the value 20 may be too high; under another, it may be reasonable. The project team should review the way the results are interpreted in every detail to evaluate interpretation accuracy. Errors may be induced due to the complexity of simulation results, especially for large scale and complex models.

Gass [1983] points out that "we do not know of any model assessment or modeling project review that indicated satisfaction with the available documentation." Nance [1977] advocates the use of standards in simulation documentation. The documentation problem should be attributed to the lack of automated support for documentation generation integrated with model development continuously throughout the entire life cycle. The model development environment [Balci 1986; Balci and Nance 1987] provides such computer-aided assistance for documenting a simulation study with respect to the phases, processes, and credibility assessment stages of the life cycle in Figure 2.

The simulation project team must devote sufficient effort in communicating technical simulation results to decision makers in a language they will understand. They must pay more attention to translating from the specialized jargon of the discipline into a form that is meaningful to the nonsimulationist and nonmodeler. Simulation results may be presented to the decision makers as integrated within a Decision Support System (DSS). With the help of a DSS, a decision maker can understand and utilize the results much better. The integration accuracy of simulation results within the DSS must be verified. If results are directly presented to the decision makers, the presentation technique (e.g., overheads, slides, films, etc.) must be ensured to be effective enough. The project management must make sure that the team members are trained and possess sufficient presentation skills.

Audit, consistency checking, desk checking, face validation, inspections, reviews, structural analysis, and walkthroughs can be applied for conducting presentation VV&T.

## 5. CONCLUDING REMARKS AND RESEARCH DIRECTIONS

The life cycle application of VV&T is extremely important for successful completion of complex and large-scale simulation studies. This point must be clearly understood by the sponsor of the simulation study and the organization conducting the simulation study. The sponsor must furnish funds under the contractual agreement and require the contractor to apply VV&T throughout the entire life cycle.

Assessing credibility throughout the life cycle of a simulation study is an onerous task. Applying the VV&T techniques throughout the life cycle is time consuming and costly. In practice, under time pressure to complete a simulation study, the VV&T and documentation are sacrificed first. Computer-aided assistance for the VV&T is required to alleviate these problems. More research is needed to bring automation to the application of the VV&T techniques.

Integration of VV&T with model development is crucial. This integration is best achieved within a computer-aided simulation software engineering environment [Balci 1986; Balci and Nance 1987]. More research is needed for this integration.

How much to test or when to stop testing depends on the study objectives. The testing should continue until we achieve sufficient confidence in credibility and acceptability of simulation results. The sufficiency of the confidence is dictated by the study objectives.

Establishing a simulation quality assurance (SQA) program within the organization conducting the simulation study is extremely important for successful credibility assessment. The SQA management structure goes beyond VV&T and is also responsible for assessing other model quality characteristics such as maintainability, reusability, and usability (human-computer interface). The management of the SQA program and the management of the simulation project must be independent of each other and neither should be able to overrule the other [Schach 1993].

Subjectivity is and will always be part of the credibility assessment for a reasonably complex simulation study. The reason for subjectivity is two-fold: modeling is an art and credibility assessment is situation dependent. A unifying approach based on the use of indicators measuring qualitative as well as quantitative aspects of a simulation study should be developed.

## 6. GLOSSARY

**Assertion Checking:** A technique for examining what *is* happening against what the modeler *assumes* is happening so as to guard model execution against potential errors.

**Audit:** A technique for investigating how adequately the simulation study is conducted with respect to established practices, standards, and guidelines and for establishing traceability within the simulation study.

**Black-Box Testing:** A technique for assessing the accuracy of model input-output transformation. It is applied by feeding inputs (test data) to the model and evaluating the corresponding outputs. The concern is how accurately the model transforms a given set of input data into a set of output data.

**Bottom-Up Testing:** A technique, used in conjunction with bottom-up model development strategy, in which testing starts with the submodels at the base level (i.e., the ones that are not decomposed further) and culminates with the submodels at the highest level. As each submodel is completed, it is thoroughly tested. When submodels belonging to the same parent have been developed and tested, the submodels are integrated and integration testing is performed. This process is repeated in a bottom-up manner until the whole model has been integrated and tested.

**Boundary Analysis:** A technique for assessing model accuracy by using test cases on the boundaries of input equivalence classes which are classes of input data that cause the model to function the same way.

**Calibration:** An iterative process in which a probabilistic characterization for an input variable or a fixed value for a parameter is tried until the model is found to be sufficiently valid.

**Cause-Effect Graphing:** A technique conducted by addressing the question of "what causes what in the model representation?" It is performed by first identifying causes and effects in the system being modeled and by examining if they are accurately reflected in the model specification. Then, the semantics are expressed in a cause-effect graph. A decision table is created by tracing back through the graph to determine combinations of causes which result in each effect. The decision table is then converted into test cases with which the model is tested.

**Communicative Model:** A model representation which can be communicated to other humans and can be judged or compared against the system and the study objectives by more than one human [Nance 1981].

**Communicative Model VV&T:** Confirming the adequacy of the communicative model to provide an acceptable level of agreement for the domain of intended application.

**Conceptual Model:** The model which is formulated in the mind of the modeler [Nance 1981].

**Consistency Checking:** A technique which deals with substantiating that: (a) the model representation does not contain contradictions, (b) the cosmetic style with which language elements (e.g., naming conventions, use of upper, lower, and mixed case, etc.) are applied is used consistently, and (c) the data elements are manipulated uniformly (e.g., data assignment to variables, data use within computations, data passing among submodels, data representation and use during model input and output).

**Data Flow Analysis:** A technique for assessing model accuracy with respect to the use of model variables. A data flow graph is constructed to aid in the data flow analysis. The nodes of the graph represent statements and corresponding variables. The edges represent control flow.

**Data VV&T:** Substantiating that each input data model has sufficient accuracy in representing the simulation model's input process (e.g., assessing the accuracy of characterizing an arrival process by Poisson probability distribution) and that all data used throughout the model development phases are accurate, complete, unbiased, and appropriate in their original and transformed forms.

**Debugging:** An iterative process the purpose of which is to uncover errors or misconceptions that cause the model's failure and to define and carry out the model changes that correct the errors.

**Design of Experiments:** The process of formulating a plan to gather the desired information at minimal cost and to enable the analyst to draw valid inferences [Shannon 1975].

**Desk Checking:** A technique for thoroughly examining one's work to ensure correctness, completeness, consistency, and unambiguity

**Domain of Applicability:** The set of prescribed conditions for which the experimental model has been tested, compared against the system to the extent possible, and judged suitable for use [Schlesinger et al. 1979].

**Domain of Intended Application:** The prescribed conditions for which the model is intended to match the system under study [Schlesinger et al. 1979].

**Execution Monitoring:** A technique for revealing errors by examining low-level information about activities and events which take place during model execution. It requires model instrumentation.

**Execution Profiling:** A technique for revealing errors by examining high-level information (profiles) about activities and events which take place during model execution. It requires model instrumentation.

**Execution Tracing:** A technique for revealing errors by "watching" the line-by-line execution of a simulation model. It requires model instrumentation.

**Experiment Design VV&T:** Substantiating that the experiments are designed and implemented with sufficient accuracy.

**Experimental Model:** The programmed model incorporating an executable description of an experiment design.

**Face Validation:** A technique in which the project team members, potential users of the model, people knowledgeable about the system under study, based on their estimates and intuition, subjectively compare model and system behaviors to judge whether the model and its results are reasonable.

**Field Testing:** A technique in which the model is placed in an operational situation for the purpose of collecting as much information as possible for model validation. It is especially useful for validating models of military combat systems.

**Formulated Problem VV&T:** Substantiating that the formulated problem contains the *actual* problem in its entirety and is sufficiently well structured to permit the derivation of a sufficiently credible solution.

**Functional Testing:** See Black-Box Testing.

**Graph-Based Analysis:** A technique for assessing model credibility based on the use of a graphical representation of the model (e.g., data flow graph, control flow graph).

**Graphical Comparisons:** A technique by which the graphs of values of model variables over time are compared with the graphs of values of system variables to investigate characteristics such as similarities in periodicities, skewness, number and location of inflection points, logarithmic rise and linearity, phase shift, trend lines, and exponential growth constants.

**Indicator:** An indirect measure of a concept which is decomposed into other indicators until the ones at the base level (i.e., the ones that are not decomposed further) are directly measurable.

**Induction:** A formal proof of correctness technique for estimating the validity of the whole set of observations based on the validity of a subset of observations as evidence.

**Inductive Assertions:** A technique for assessing model correctness based on an approach that is very close to formal proof of model correctness. It is conducted in three steps: (1) input-to-output relations for all model variables are identified; (2) these relations are converted into assertion statements and are placed along the model execution paths in such a way as to divide the model into a finite number of "assertion-bound" paths, i.e., an assertion statement lies at the beginning and end of each model execution path; and (3) verification is achieved by proving that for each path: if the assertion at the beginning of the path is true, and all statements along the path are executed, then the assertion at the end of the path is true. If all paths plus model termination can be proved, by induction, the model is proved to be correct.

**Inference:** A formal proof of correctness technique for deriving logical conclusions from the premises given.

**Input Data Model VV&T:** Substantiating that the input data model has sufficient accuracy in representing the simulation model's input process (e.g., assessing the accuracy of characterizing an arrival process by Poisson probability distribution).

**Inspections:** A technique conducted by a team of four to six members with the objective of finding and documenting faults. It consists of five distinct phases: overview, preparation, inspection, rework, and follow-up.

**Lamda Calculus:** A technique for transforming the model into formal expressions by which mathematical proof of correctness techniques can be applied for the purpose of VV&T.

**Level of Agreement:** The required correspondence between the model and the system, consistent with the domain of intended application and the study objectives [Schlesinger et al. 1979].

**Logical Deduction:** A formal proof of correctness technique for reasoning in which a conclusion follows necessarily from the premises given.

**Model:** Representation and abstraction of anything such as a system, concept, problem, or phenomena.

**Model Builder's Risk:** The probability of committing type I error.

**Model Certification:** Confirmation (usually by a third party) that a simulation model, within its domain of applicability, can produce results which are sufficiently credible with respect to the study objectives.

**Model Instrumentation:** The insertion of additional code (probes) into the executable model for the purpose of collecting information about model behavior during execution. Probe locations are determined manually or automatically based on static analysis of model structure. Automated instrumentation is accomplished by a preprocessor which analyzes the model static structure (usually via graph-based analysis) and inserts probes at appropriate places.

**Model Qualification:** Justifying that all assumptions underlying the conceptual model are appropriate and the conceptual model provides an adequate representation of the system under study with respect to the study objectives.

**Model Range of Accuracy:** Simultaneous confidence intervals for the differences between the means of corresponding model and system output variables obtained by running the model with the "same" input data that drives the real system.

**Model Testing:** Demonstrating that inaccuracies exist or revealing the existence of errors in the model. In model testing, we subject the model to test data or test cases to see if it functions properly. "Test failed" implies the failure of the model, not the test.

**Model User's Risk:** The probability of committing type II error.

**Model Validation:** Substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives. Model validation deals with building the *right* model. It is conducted by running the model under the “same” input conditions that drive the system and by comparing model behavior with the system behavior.

**Model Verification:** Substantiating that the model is transformed from one form into another, as intended, with sufficient accuracy. Model verification deals with building the model *right*. The accuracy of transforming a problem formulation into a model specification or the accuracy of converting a model representation in micro flowchart into an executable computer program is evaluated in model verification.

**Model VV&T:** Model testing is conducted to perform model validation and verification. Some tests are devised to evaluate the behavioral accuracy (i.e., validity) of the model, and some tests are intended to judge the accuracy of model transformation from one form into another (verification). Therefore, we commonly refer to the whole process as model VV&T.

**Partition Analysis:** A technique for testing the model with the test data generated by analyzing the model’s functional representatives (partitions). It is accomplished by: (1) decomposing both model specification and implementation into functional representatives (partitions), (2) comparing the elements and prescribed functionality of each partition specification with the elements and actual functionality of corresponding partition implementation, (3) deriving test data to extensively test the functional behavior of each partition, and (4) testing the model by using the generated test data.

**Path Analysis:** A technique for assessing model correctness on the basis of complete testing of all model control paths. It is performed in three steps: (1) the model control structure is determined and represented in a control flow diagram, (2) test data is generated to cause selected model logical paths to be executed, and (3) by using the generated test data, the model is forced to proceed through each path in its execution structure, thereby providing comprehensive testing.

**Predicate Calculus:** A technique that provides rules for manipulating predicates. A predicate is a combination of simple relations which will either be true or false. The model can be defined in terms of predicates and manipulated using the rules of the predicate calculus for the purpose of VV&T.

**Predicate Transformation:** A technique that provides a basis for verifying model correctness by formally defining the semantics of the model with a mapping which transforms model output states to all possible model input states. This representation provides the basis for proving model correctness.

**Predictive Validation:** A technique by which model validation is conducted using past system data. The model is driven by past system input data and its forecasts are compared with the corresponding past system output data to test the predictive ability of the model.

**Presentation VV&T:** Substantiating that the simulation results are interpreted, documented, and communicated with sufficient accuracy.

**Programmed Model:** A model representation that admits execution by a computer to produce results [Nance 1981].

**Programmed Model VV&T:** Substantiating that the programmed model possesses sufficient accuracy in representing the system under study.

**Proof of Correctness:** A technique for expressing the model in a precise notation and then mathematically proving that the executed model terminates and it satisfies the requirements specification with sufficient accuracy.

**Regression Testing:** A technique for substantiating that correcting errors and/or making changes in the model do not create other errors and adverse side-effects. It is usually accomplished by retesting the modified model with the previous test data sets used.

**Reviews:** A technique conducted by a team of experts and managers with the objective of finding and documenting faults.

**Self-Driven Simulation Model:** A model driven by input values obtained via sampling from probability distributions using random numbers.

**Semantic Analysis:** A technique by which the simulation programming language compiler generates a wealth of information to help the modeler determine if the true intent is accurately translated into the executable code.

**Sensitivity Analysis:** A technique for systematically changing the values of model input variables and parameters over some range of interest and observing the effect upon model behavior [Shannon 1975]. Unexpected effects may reveal invalidity.

**Simulation:** The process of constructing a model of a system which contains a problem and conducting experiments with the model on a computer for a specific purpose of experimentation to solve the problem.

**Simulation Quality Assurance:** Refers to the management structure responsible for planning, preparing test cases, and administering VV&T activities throughout the life cycle of a simulation study to assure sufficient credibility and acceptability of simulation study results.

**Stress Testing:** A technique for assessing model validity under extreme workload conditions. This is usually accomplished by increasing the congestion in the model.

**Structural Analysis:** A technique for examining the model structure and determining if it adheres to structured principles. It is conducted by constructing a control flow graph of the model structure and examining the graph for anomalies, such as multiple entry and exit points, excessive levels of nesting within a structure, and questionable practices such as the use of unconditional branches (i.e., GOTOs).

**Submodel Testing:** The experimental model is instrumented to collect data on all input and output variables of a submodel. The system is similarly instrumented (if possible) to collect similar data. Then, each submodel behavior is compared with corresponding subsystem behavior to judge submodel validity. If a subsystem can be modeled analytically (e.g., as an M/M/1 model), its exact solution can be compared against the simulation solution to assess validity quantitatively.

**Symbolic Debugging:** A technique which allows the modeler to locate errors and check numerous circumstances which lead up to the errors by employing a debugging tool that allows the modeler to manipulate model execution while viewing the model at the source code level. By setting "breakpoints", the modeler can interact with the entire model one step at a time, at predetermined locations, or under specified conditions.

**Symbolic Execution:** A technique for assessing model accuracy by executing the model using symbolic values rather than actual data values for input. It is performed by feeding symbolic inputs into the (sub)model and producing expressions for the output which are derived from the transformation of the symbolic data along model execution paths.

**Syntax Analysis:** A technique carried on by the simulation programming language compiler to assure that the mechanics of the language are applied correctly.

**System and Objectives Definition VV&T:** Substantiating that the system characteristics are identified and the study objectives are explicitly defined with sufficient accuracy.

**Top-Down Testing:** A technique, used in conjunction with top-down model development strategy, in which testing starts with the submodels at the highest level and culminates with the submodels at the base level (i.e., the ones that are not decomposed further). As each submodel is completed, it is thoroughly tested. When submodels belonging to the same parent have been developed and tested, the submodels are integrated and integration testing is performed. This process is repeated in a top-down manner until the whole model has been integrated and tested.

**Trace-Driven Simulation Model:** A model driven by input sequences extracted from trace data obtained through measurement of the real system.

**Turing Test:** A technique in which people with expert knowledge about the system under study are presented with two sets of output data obtained, one from the model and one from the system, under the same input conditions. Without identifying which one is which, the people are asked to differentiate between the two. If they succeed, they are asked how they were able to do it. Their response provides valuable feedback for correcting model representation. If they cannot differentiate, our confidence in model validity is increased.

**Type I Error:** The error of rejecting the model credibility when in fact the model is sufficiently credible. Occurrence of type I error may increase the cost of model development or may cause the simulation study to end unsuccessfully.

**Type II Error:** The error of accepting the model credibility when in fact the model is *not* sufficiently credible. Consequences of committing type II error can be catastrophic.

**Type III Error:** The error of solving the wrong problem. Once the Type III error is committed, regardless of how well the problem is solved, the simulation study will either end unsuccessfully or with the Type II error.

**Visualization:** Displaying graphical images of internal and external dynamic behavior of a simulation model during execution.

**Walkthroughs:** A technique conducted by a team composed of a coordinator, model developer, and three to six other members with the objective of discovering and documenting faults.

**White-Box Testing:** A technique which employs data flow and control flow diagrams to assess the accuracy of internal model structure by examining model elements such as internal logic, internal data representations, submodel interfaces, and model execution paths. White-box testing is quite effective for detecting redundant code, faulty model structure, and special case errors.

## REFERENCES


Ackerman, A.F., P.J. Fowler, and R.G. Ebenau (1983), "Software Inspections and the Industrial Production of Software," In *Software Validation: Inspection, Testing, Verification, Alternatives*, Proceedings of the Symposium on Software Validation (Darmstadt, FRG, Sept. 25-30), Hans-Ludwig Hausen, Ed., pp. 13-40.

Adrion, W.R., M.A. Branstad, and J.C. Cherniavsky (1982), "Validation, Verification, and Testing of Computer Software," *Computing Surveys 14*, 2 (June), 159-192.

Aigner, D.J. (1972), "A Note on Verification of Computer Simulation Models," *Management Science 18*, 11 (Nov.), 615-619.

Allen, F.E. and J. Cocke (1976), "A Program Data Flow Analysis Procedure," *Communications of the ACM 19*, 3 (Mar.), 137-147.

Backhouse, R.C. (1986), *Program Construction and Verification*, Prentice-Hall International (UK) Ltd., London, Great Britain.

Balci, O. (1986), "Requirements for Model Development Environments," *Computers & Operations Research 13*, 1, 53-67.

Balci, O. (1988), "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-level Languages," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 287-295.

Balci, O. (1990), "Guidelines for Successful Simulation Studies," In *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R.P. Sadowski, and R.E. Nance, Eds. IEEE, Piscataway, NJ, 25-32.

Balci, O. (1994), "Principles of Simulation Model Validation, Verification, and Testing," Technical Report TR-94-24, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Balci, O. and R.E. Nance (1985), "Formulated Problem Verification as an Explicit Requirement of Model Credibility," *Simulation 45*, 2, 76-86.

Balci, O. and R.E. Nance (1987), "Simulation Model Development Environments: A Research Prototype," *Journal of Operational Research Society 38,* 8, 753-763.

Balci, O. and R.G. Sargent (1981), "A Methodology for Cost-Risk Analysis in the Statistical Validation of Simulation Models," *Communications of the ACM 24*, 4, 190-197.

Balci, O. and R.G. Sargent (1982a), "Some Examples of Simulation Model Validation Using Hypothesis Testing," In *Proceedings of the 1982 Winter Simulation Conference,* H.J. Highland, Y.W. Chao, and O.S. Madrigal, Eds. IEEE, Piscataway, 620-629.

Balci, O. and R.G. Sargent (1982b), "Validation of Multivariate Response Models Using Hotelling's Two-Sample $T^2$ Test," *Simulation 39*, 6 (Dec.), 185-192.

Balci, O. and R.G. Sargent (1983), "Validation of Multivariate Response Trace-Driven Simulation Models," In *Performance '83,* A K. Agrawala and S.K. Tripathi, Eds. North-Holland, Amsterdam, 309-323.

Balci, O. and R.G. Sargent (1984), "Validation of Simulation Models via Simultaneous Confidence Intervals," *American Journal of Mathematical and Management Sciences 4*, 3&4, 375-406.

Banks, J. and J.S. Carson (1984), *Discrete-Event System Simulation,* Prentice-Hall, Englewood Cliffs, NJ.

Banks, J., D. Gerstein, and S.P. Searles (1987), "Modeling Processes, Validation, and Verification of Complex Simulations: A Survey," In *Methodology and Validation*, O. Balci, Ed. SCS, San Diego, CA, 13-18.

Barendregt, H.P. (1981), *The Lambda Calculus: Its Syntax and Semantics*, North-Holland, New York, N.Y.

Chusho, T. (1987), "Test Data Selection and Quality Estimation Based on the Concept of Essential Branches for Path Testing," *IEEE Transactions on Software Engineering SE-13*, 5 (May), 509-517.

Cohen, K.J. and R.M. Cyert (1961), "Computer Models in Dynamic Economics," *Quarterly Journal of Economics 75*, 1 (Feb.), 112-127.

Damborg, M.J. and L.F. Fuller (1976), "Model Validation Using Time and Frequency Domain Error Measures," ERDA Report 76-152, NTIS, Springfield, VA.

Deutsch, M.S. (1982), *Software Verification and Validation: Realistic Project Approaches*, Prentice-Hall, Englewood Cliffs, NJ.

Dijkstra, E.W. (1975), "Guarded Commands, Non-determinacy and a Calculus for the Derivation of Programs," *Communications of the ACM 18*, 8 (Aug.), 453-457.

Dillon, L.K. (1990), "Using Symbolic Execution for Verification of Ada Tasking Programs," *ACM Transactions on Programming Languages and Systems 12*, 4, 643-669.

Dobbins, J.H. (1987), "Inspections as an Up-Front Quality Technique," In *Handbook of Software Quality Assurance*, G.G. Schulmeyer and J.I. McManus, Eds. Van Nostrand-Reinhold Company, New York, NY, pp. 137-177.

Dunn, R.H. (1987), "The Quest for Software Reliability," In *Handbook of Software Quality Assurance*, G.G. Schulmeyer and J.I. McManus, Eds. Van Nostrand-Reinhold Company, New York, NY, pp. 342-384.

Elmaghraby, S.E. (1968), "The Role of Modeling in IE Design," *Industrial Engineering 19*, 6, 292-305.

Emshoff, J.R. and R.L. Sisson (1970), *Design and Use of Computer Simulation Models*, MacMillan, New York, NY.

Fairley, R.E. (1975), "An Experimental Program-Testing Facility," *IEEE Transactions on Software Engineering SE-1*, 4, 350-357.

Fairley, R.E. (1976), "Dynamic Testing of Simulation Software," In *Proceedings of the 1976 Summer Computer Simulation Conference* (Washington, D.C., July 12-14), "Simulation Councils, La Jolla, CA, 708-710.

Fishman, G.S. (1978), *Principles of Discrete Event Simulation,* Wiley-Interscience, New York, NY.

Fishman, G.S. and P.J. Kiviat (1967), "The Analysis of Simulation Generated Time Series," *Management Science 13*, 7 (July), 525-557.

Forrester, J.W. (1961), *Industrial Dynamics*, MIT Press, Cambridge, MA.

Gafarian, A.V. and J.E. Walsh (1969), "Statistical Approach for Validating Simulation Models by Comparison with Operational Systems," In *Proceedings of the 4th International Conference on Operations Research*, John Wiley & Sons, New York, NY, 702-705.

Gallant, A.R., T.M. Gerig, and J.W. Evans (1974), "Time Series Realizations Obtained According to an Experimental Design," *J. American Statistical Association 69*, 347 (Sept.), 639-645.

Garratt, M. (1974), "Statistical Validation of Simulation Models," In *Proceedings of the 1974 Summer Computer Simulation Conference* (Houston, Tex., July 9-11). Simulation Councils, La Jolla, CA, 915-926.

Gass, S.I. (1983), "Decision-Aiding Models: Validation, Assessment, and Related Issues for Policy Analysis," *Operations Research 31*, 4, 603-631.

Hermann, C.F. (1967), "Validation Problems in Games and Simulations with Special Reference to Models of International Politics," *Behavioral Science 12*, 3 (May), 216-231.

Hetzel, W. (1984), *The Complete Guide to Software Testing*, QED Information Sciences, Wellesley, MA.

Hollocker, C.P. (1987), "The Standardization of Software Reviews and Audits," In *Handbook of Software Quality Assurance*, G.G. Schulmeyer and J.I. McManus, Eds. Van Nostrand-Reinhold Company, New York, NY, pp. 211-266.

Howden, W.E. (1976), "Reliability of the Path Analysis Testing Strategy," *IEEE Transactions on Software Engineering SE-2*, 3 (Sept.), 208-214.

Howden, W.E. (1980), "Functional Program Testing," *IEEE Transactions on Software Engineering SE-6*, 2, 162-169.

Howrey, P. and H.H. Kelejian (1969), "Simulation Versus Analytical Solutions," In *The Design of Computer Simulation Experiments,* T.H. Naylor, Ed. Duke University Press, Durham, NC, 207-231.

Hunt, A.W. (1970), "Statistical Evaluation and Verification of Digital Simulation Models Through Spectral Analysis," Ph.D. Dissertation, University of Texas at Austin, Austin, TX.

Jacobson, S.H. and E. Yücesan (1993), "On the NP-Completeness of Verifying Structural Properties of Discrete Event Simulation Models," Technical Report, Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA.

Khanna, S. (1991), "Logic Programming for Software Verification and Testing," *The Computer Journal 34*, 4, 350-357.

Kheir, N.A. and W.M. Holmes (1978), "On Validating Simulation Models of Missile Systems," *Simulation 30*, 4 (Apr.), 117-128.

King, J.C. (1976), "Symbolic Execution and Program Testing," *Communications of the ACM 19*, 7 (July), 385-394.

Kleijnen, J.P.C. (1975), *Statistical Techniques in Simulation*, Volume 2, Marcel Dekker, New York, NY.

Knepell, P.L. and D.C. Arangno (1993), *Simulation Validation: A Confidence Assessment Methodology*, Monograph 3512-04, IEEE Computer Society Press, Los Alamitos, CA.

Knight, J.C. and E.A. Myers (1993), "An Improved Inspection Technique," *Communications of the ACM 36*, 11, 51-61.

Law, A.M. (1983), "Statistical Analysis of Simulation Output Data," *Operations Research 31*, 6, 983-1029.

Law, A.M. and W.D. Kelton (1991), *Simulation Modeling and Analysis,* Second Edition, McGraw-Hill, New York, NY.

Manna, Z., S. Ness, and J. Vuillemin (1973), "Inductive Methods for Proving Properties of Programs," *Communications of the ACM 16*, 8 (Aug.), 491-502.

Martin, J. and C. McClure (1985), *Diagramming Techniques for Analysts and Programmers,* Prentice-Hall, Englewood Cliffs, NJ.

Miller, D.K. (1975), "Validation of Computer Simulations in the Social Sciences," In *Proceedings of the Sixth Annual Conference on Modeling and Simulation* (Pittsburg, PA), pp. 743-746.

Miller, D.R. (1974a), "Model Validation Through Sensitivity Analysis," In *Proceedings of the 1974 Summer Computer Simulation Conference* (Houston, TX, July 9-11). Simulation Councils, La Jolla, CA, 911-914.

Miller, D.R. (1974b), "Sensitivity Analysis and Validation of Simulation Models," *J. Theoretical Biology 48*, 2 (Dec.), 345-360.

Moose, R.L. and R.E. Nance (1989), "The Design and Development of an Analyzer for Discrete Event Model Specifications," In *Impacts of Recent Computer Advances on Operations Research*, R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart, Eds. Elsevier, New York, NY, 407-421.

Myers, G.J. (1978), "A Controlled Experiment in Program Testing and Code Walkthroughs/ Inspections," *Communications of the ACM 21*, 9 (Sept.), 760-768.

Myers, G.J. (1979), *The Art of Software Testing*, John Wiley & Sons, New York, N.Y.

Nance, R.E. (1977), "The Feasibility of and Methodology for Developing Federal Documentation Standards for Simulation Models: Final Report to the National Bureau of Standards," Department of Computer Science, VPI&SU, Blacksburg, Va., June.

Nance, R.E. (1981), "Model Representation in Discrete Event Simulation: The Conical Methodology," Technical Report CS81003-R, Department of Computer Science, VPI&SU, Blacksburg, VA.

Nance, R.E. (1987), "The Conical Methodology: A Framework for Simulation Model Development," In *Methodology and Validation*, O. Balci, Ed. SCS, San Diego, CA, 38-43.

Nance, R.E. (1994), "Conical Methodology: An Evolutionary Convergence of Systems and Software Engineering" *Annals of Operations Research*, to appear.

Nance, R.E. and C.M. Overstreet (1987), "Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications," *Transactions of the SCS 4*, 1 (Jan.), 33-57.

Naylor, T.H. and J.M. Finger (1967), "Verification of Computer Simulation Models," *Management Science 14*, 2 (Feb.), B92-B101.

Ould, M.A. and C. Unwin (1986), *Testing in Software Development*, Cambridge University Press, Great Britain.

Overstreet, C.M. and R.E. Nance (1985), "A Specification Language to Assist in Analysis of Discrete Event Simulation Models," *Communications of the ACM 28*, 2, 190-201.

Ören, T.I. (1981), "Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference," *Communications of the ACM 24*, 4, 180-189.

Ören, T.I. (1986), "Artificial Intelligence in Quality Assurance of Simulation Studies," In *Modelling and Simulation Methodology in the Artificial Intelligence Era*, M.S. Elzas, T.I. Ören, and B.P. Zeigler, Eds. North Holland, Amsterdam, pp. 267-278.

Ören, T.I. (1987), "Quality Assurance Paradigms for Artificial Intelligence in Modelling and Simulation," *Simulation 48*, 4 (Apr.), 149-151.

Paul, R.J. (1989), "Visual Simulation: Seeing is Believing?" In *Impacts of Recent Computer Advances on Operations Research*, R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart, Eds. Elsevier, New York, NY, 422-432.

Prather, R.E. and J.P. Myers, Jr. (1987), "The Path Prefix Software Testing Strategy," *IEEE Transactions on Software Engineering SE-13*, 7 (July), 761-766.

Ramamoorthy, C.V., S.F. Ho, and W.T. Chen (1976), "On the Automated Generation of Program Test Data," *IEEE Transactions on Software Engineering SE-2*, 4 (Dec.), 293-300.

Reynolds, C. and R.T. Yeh (1976), "Induction as the Basis for Program Verification," *IEEE Transactions on Software Engineering SE-2*, 4, 244-252.

Richardson, D.J. and L.A. Clarke (1985), "Partition Analysis: A Method Combining Testing and Verification," *IEEE Transactions on Software Engineering SE-11*, 12 (Dec.), 1477-1490.

Rowland, J.R. and W.M. Holmes (1978), "Simulation Validation with Sparse Random Data," *Computers and Electrical Engineering 5*, 3 (Mar.), 37-49.

Sargent, R.G. (1992), "Validation and Verification of Simulation Models," In *Proceedings of the 1992 Winter Simulation Conference*, J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, Eds. IEEE, Piscataway, NJ, 104-114.

Satterthwaite, E. (1972), “Debuggine Tools for High Level Languages,” *Software – Practice and Experience 2*, 3, 197-217.

Schach, S.R. (1993), *Software Engineering*, Second Edition, Irwin, Homewood, IL.

Schlesinger, S., et al. (1979), “Terminology for Model Credibility,” *Simulation 32*, 3, 103-104.

Schmeiser, B. (1981), “Random Variate Generation,” In *Proceedings of the 1981 Winter Simulation Conference*, T.I. Ören, C.M. Delfosse, and C.M. Shub, Eds. IEEE, Piscataway, NJ, 227-242.

Schriber, T.J. (1974), *Simulation Using GPSS*, John Wiley and Sons, New York, NY.

Schruben, L.W. (1980), “Establishing the Credibility of Simulations,” *Simulation 34*, 3 (Mar.), 101-105.

Shannon, R.E. (1975), *Systems Simulation: The Art and Science*, Prentice-Hall, Englewood Cliffs, NJ.

Stucki, L.G. (1977), “New Directions in Automated Tools for Improving Software Quality,” In *Current Trends in Programming Methodology*, Vol. 2, R. Yeh, Ed. Prentice-Hall, Englewood Cliffs, NJ, pp. 80-111.

Teorey, T.J. (1975), “Validation Criteria for Computer System Simulations,”*Simuletter 6*, 4 (July), 9-20.

Theil, H. (1961), “*Economic Forecasts and Policy,* North-Holland, Amsterdam, The Netherlands.

Turing, A.M. (1963), “Computing Machinery and Intelligence,” In *Computers and Thought,* E.A. Feigenbaum and J. Feldman, Eds. McGraw-Hill, New York, NY, 11-15.

Tytula, T.P. (1978), “A Method for Validating Missile System Simulation Models,” Technical Report E-78-11, U.S. Army Missile R&D Command, Redstone Arsenal, AL, June.

U.S. GAO (1987), *DOD Simulations: Improved Assessment Procedures Would Increase the Credibility of Results*, U.S. General Accounting Office GAO/PEMD-88-3, Washington, DC, Dec.

Van Horn, R.L. (1971), “Validation of Simulation Results,” *Management Science 17*, 5 (May), 247-258.

Watts, D. (1969), “Time Series Analysis,” In *The Design of Computer Simulation Experiments,* T.H. Naylor, Ed. Duke University Press, Durham, NC, 165-179.

Whitner, R.B. and O. Balci (1989), “Guidelines for Selecting and Using Simulation Model Verification Techniques,” In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 559-568.

Wilson, J.R. and A.A.B. Pritsker (1978), “A Survey of Research on the Simulation Startup Problem,” *Simulation 31*, 2, 55-58.

Woolley, R.N. and M. Pidd (1981), “Problem Structuring – A Literature Review,” *Journal of Operational Research Society 32*, 3, 197-206.

Wright, R.D. (1972), “Validating Dynamic Models: An Evaluation of Tests of Predictive Power,” In *Proceedings of the 1972 Summer Computer Simulation Conference* (San Diego, CA, July 14-16). Simulation Councils, La Jolla, CA, 1286-1296.

Yeh, R.T. (1977), “Verification of Programs by Predicate Transformation,” In *Current Trends in Programming Methodology*, Vol. 2, R. Yeh, Ed. Prentice-Hall, Englewood Cliffs, NJ, pp. 228-247.

Yourdon, E. (1985), *Structured Walkthroughs*, 3rd Edition, Yourdon Press, New York, NY.

Yücesan, E. and S.H. Jacobson (1992), “Building Correct Simulation Models is Difficult,” In *Proceedings of the 1992 Winter Simulation Conference,* J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, Eds. IEEE, Piscataway, NJ, 783-790.

# VALIDATION, VERIFICATION, AND TESTING TECHNIQUES THROUGHOUT THE LIFE CYCLE OF A SIMULATION STUDY


OSMAN BALCI

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106, U.S.A.

balci@vt.edu



1 January 1994

Revised: 27 June 1994

**To appear in:** *Annals of Operations Research*, December 1994.

## ABSTRACT

Life cycle validation, verification, and testing (VV&T) is extremely important for the success of a simulation study. This paper surveys current software VV&T techniques and current simulation model VV&T techniques and describes how they can all be applied throughout the life cycle of a simulation study. The processes and credibility assessment stages of the life cycle are described and the applicability of the VV&T techniques for each stage is stated. A glossary is provided to explicitly define important terms and VV&T techniques.

**CR Categories and Subject Descriptors:** I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event;* D.2.4 [**Software Engineering**]: Program Verification—*Validation;* D.2.5 [**Software Engineering**]: Testing and Debugging

**Additional Key Words and Phrases:** Credibility assessment of simulations, life cycle of a simulation study, principles of model assessment, simulation quality assurance.

## TABLE OF CONTENTS