CHAPTER 10

Verification, Validation, and Testing

OSMAN BALCI
Virginia Polytechnic Institute and State University

10.1 INTRODUCTION

A simulation study is conducted for a variety of purposes, including problem solving and training. Starting with problem formulation and culminating with presentation of simulation study results, it consists of complex processes of formulation, analysis, modeling, and experimentation (or exercise). A typical simulation study requires multifaceted knowledge in diverse disciplines such as operations research, computer science, statistics, and engineering. Due to the complex processes and multifaceted knowledge requirements, simulation practitioners and managers face significant technical challenges in conducting *successful* simulation studies. A successful simulation study is defined to be the one that produces a sufficiently credible solution that is accepted and used by the decision makers.

To increase significantly the probability of success in conducting a simulation study, an organization must have a department or group called *simulation quality assurance* (SQA). The SQA group is responsible for total quality management and works closely with the simulation project managers in planning, preparing, and administering quality assurance activities throughout the simulation study. The SQA is a managerial approach that is critically essential for the success of a simulation study. Ören [1–3] presents concepts, criteria, and paradigms that can be used in establishing an SQA program within an organization.

Assuring total quality involves the measurement and assessment of a variety of quality characteristics such as accuracy, execution efficiency, maintainability, portability, reusability, and usability (human–computer interface). Simulation study objectives dictate a priority ordering of these quality characteristics since all of them cannot be achieved at the same level.

The purpose of this chapter is to present principles and techniques for the assessment of accuracy throughout the life cycle of a simulation study. The accuracy quality characteristic is assessed by conducting verification, validation, and testing (VV&T).

Handbook of Simulation, Edited by Jerry Banks.
ISBN 0-471-13403-1

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 a micro flowchart into an executable computer program is evaluated in model verification.

Model validation is substantiating that within its domain of applicability, the model behaves with satisfactory accuracy consistent with the study objectives. Model validation deals with building the *right* model.

An activity of accuracy assessment can be labeled as verification or validation based on an answer to the following question: In assessing the accuracy, is the model behavior compared with respect to the corresponding system behavior through mental or computer execution? If the answer is "yes," model validation is conducted; otherwise, it implies that the transformational accuracy is judged implying model verification.

Model testing is ascertaining whether inaccuracies or errors exist in the model. In model testing, the model is subjected to test data or test cases to determine if it functions properly. "Test failed" implies the failure of the model, not the test. A test is devised and testing is conducted to perform either validation or verification or both. 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, the entire process is commonly called *model VV&T*.

Testing should not be interpreted just as functional testing which requires computer execution of the model. Administering reviews, inspections, and walkthroughs is similar to devising a test under which model accuracy is judged. In this case, panel members become part of the devised test and the testing is conducted by each member executing a set of tasks. Therefore, informal techniques described in Section 10.4 are also considered testing techniques.

10.2 LIFE CYCLE AND A CASE STUDY

The processes and phases of the life cycle of a simulation study and a simulation and modeling case study are presented in this section. The case study is used throughout the chapter to illustrate the life cycle and the VV&T principles and techniques. The life cycle of a simulation study is presented in Figure 10.1 [4, 5]. The phases are shown by shaded oval symbols. The dashed arrows describe the processes that relate the phases to each other. The solid arrows refer to the credibility assessment stages. Banks et al. [6] and Knepell and Arangno [7] 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 10 processes of the life cycle are shown by the dashed arrows in Figure 10.1. 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 10 processes.

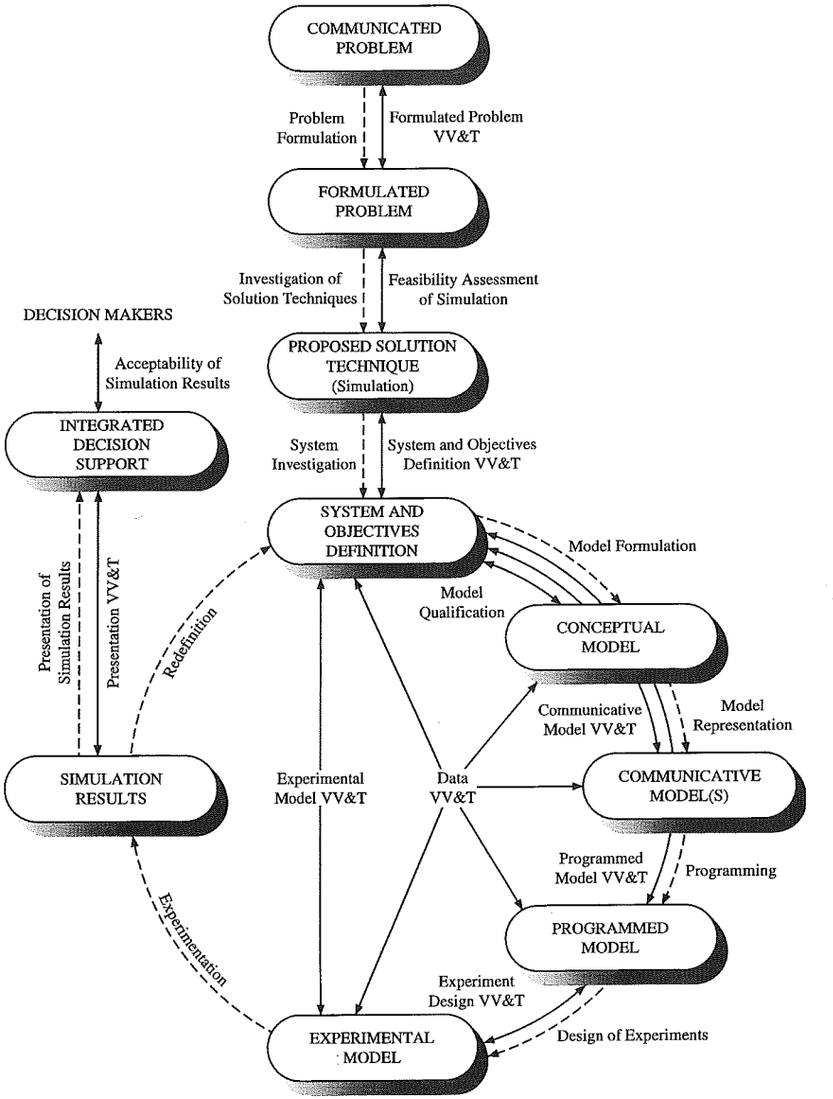

Figure 10.1 Life cycle of a simulation study.

10.2.1 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, contractor, or a consultant/research group). The problem communicated 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 [8].

Balci and Nance [9] 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.

Case Study. The town of Blacksburg in Virginia (client) receives complaints from the drivers using the traffic intersection at Prices Fork Road and Toms Creek Road, shown in Figure 10.2, about too much waiting during rush-hour periods. The town hires a consulting company (the contractor) to conduct a study and propose a solution to the problem.

The contractor conducts the process of problem formulation and determines the study objective as follows:

> Identify which operating policy should be implemented at the traffic intersection so as to reduce the average waiting time of vehicles in each travel path to an acceptable level during rush-hour periods. Possible operating policies include different light timings, two-way stop signs, four-way stop signs, flashing red and yellow lights, and constructional changes such as adding new lanes.

10.2.2 Investigation of Solution Techniques

All alternative techniques that can be used to solve the formulated problem should be identified. A technique whose solution is estimated to be too costly or is judged not to be 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 that 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 problem communicated is formulated with 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, it is assumed here 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 preceeding in the life cycle.

Case Study. The contractor investigates all possible solution techniques and selects discrete-event simulation as the one with the highest benefits/cost ratio.

10.2.3 System Investigation

Characteristics of the system that contains the problem formulated should be investigated for consideration in system definition and modeling. Shannon [10] identifies six

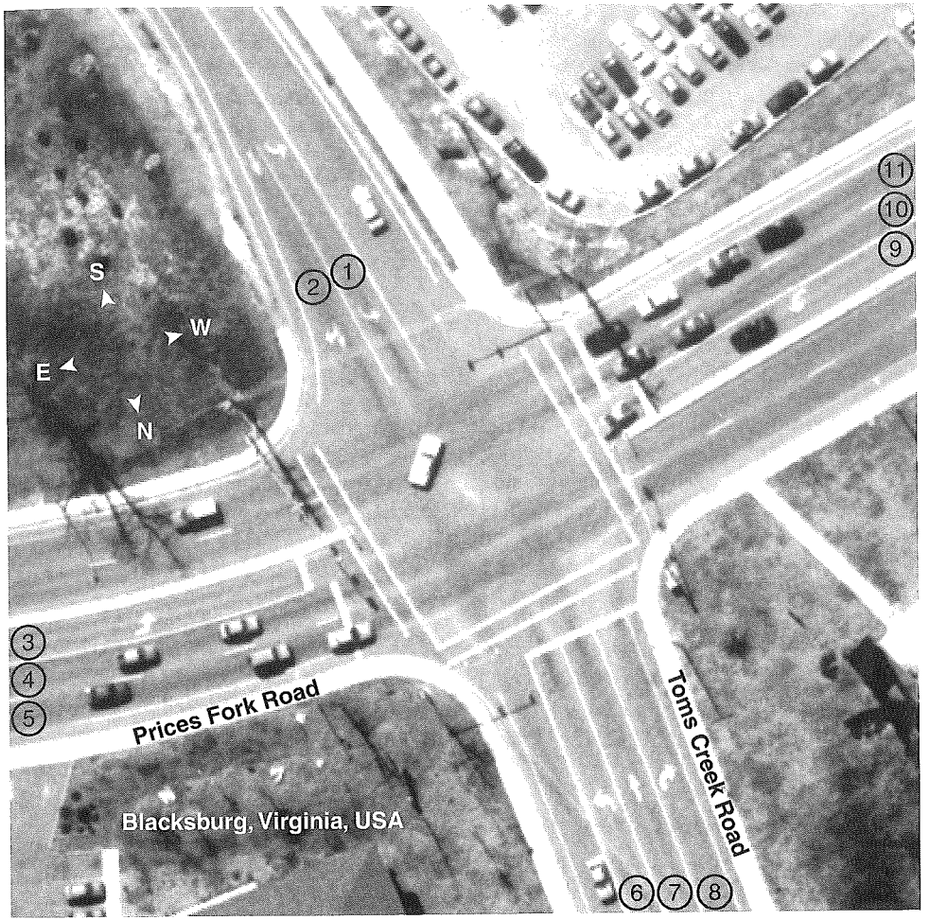

Figure 10.2 Traffic intersection at Prices Fork Road and Toms Creek Road.

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 deal primarily 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 affect its state significantly. The input variables are identified by assessing the significance of their influence on the system's state with regard to the study objectives. Underestimating the influence of an input variable may result in inaccurate environment definition.

Some complex systems may show *counterintuitive behavior*, which should be identified 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 [10]. To be able to identify counterintuitive behavior, it is essential that the simulation project employ 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.

The *interdependency* and *organization* characteristics of the system should be examined prior to the abstraction of the real system for the purpose of modeling. 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 whose complexity is manageable and the system characteristics are documented, model formulation process can be started following the system and objectives definition VV&T.

Case Study. The contractor conducts the process of system investigation. It is determined that the traffic intersection will not change during the course of the study. The interarrival time of vehicles in lane L_j where $j = 1, 2, 3, \ldots, 11$ is identified as an input variable making up the environment, whereas pedestrians, emergency vehicles, and bicycles are excluded from the system definition due to their negligible effect on the system's state. No counterintuitive behavior can be identified. The system performance does not deteriorate over time.

10.2.4 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 that is formulated in the mind of the modeler [5]. Model formulation and model representation constitute the process of model design.

Input data analysis and modeling [11, 12] 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 that 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 traffic intersection), we collect data on an input random variable (e.g., interarrival time of vehicles), 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 [13]. In a trace-driven simulation, we trace the system (e.g., using hardware and software monitors) and

TABLE 10.1 Probabilistic Models of Vehicle Interarrival Times for Each Lane

Lane Number(s)	Probability Distribution	Location Parameter	Scale Parameter	Shape Parameter
1, 2	Gamma	0.06494	4.05843	1.1031
3	Weibull	1.99415	31.1737	0.79453
4	Weibull	0	8.71858	0.8773
5	Lognormal	0	1.42075	1.40056
6	Weibull	0	32.9441	1.14441
7	Weibull	0.99627	27.6663	0.70053
8	Weibull	0	33.1788	1.46385
9	Lognormal	0	1.93024	1.11273
10	Weibull	0	6.91658	0.78088
11	Weibull	0	5.57763	0.71616

utilize the refined trace data as the input data model to use in driving the simulation model.

Case Study. All assumptions made in abstracting the traffic intersection operation under the study objective are stated and listed explicitly. Data are collected on the interarrival times of vehicles, current traffic light timing, probabilities of turns, and travel times in each travel path. A single arrival process is observed for lanes 1 and 2 and divided probabilistically. ExpertFit software [13] is used to identify probabilistic models of the input process. The results of input data modeling are given in Table 10.1.

The estimated probabilities of right turns are presented in Table 10.2. The probability distributions identified characterize the rush-hour traffic conditions and are used to sample from in driving the self-driven simulation model built.

10.2.5 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" [5]. A communicative model (i.e., a simulation model design specification) may be represented in any of the following forms: (1) structured, computer-assisted graphs, (2) flowcharts, (3) structured English and pseudocode, (4) entity-cycle

TABLE 10.2 Estimated Probabilities of Right Turns

Location	Probability
Turning to lane 2 from the combined arrival process for lanes 1 and 2	0.634
Right turn in lane 2	0.346
Right turn in lane 5	0.160
Right turn in lane 11	0.140

(or activity-cycle) diagrams, (5) condition specification [14], and (6) more than a dozen diagramming techniques described in ref. 15.

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 on (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.

Case Study. The Visual Simulation Environment (VSE) software product [16–18] is selected for simulation model development and experimentation. An aerial photograph of the traffic intersection obtained from the town of Blacksburg is scanned as shown in Figure 10.2. Vehicle images, tree branches, and light posts on the roads are removed from the scanned image using Adobe Photoshop software. The cleaned image is brought into the VSE Editor by clicking, dragging, and dropping. The photographic image is decomposed into components represented as circles as shown in Figure 10.3. The components are connected with each other using the path tool. The traffic light for each lane is depicted by a line which changes color during animation. New classes are created by inheriting from the built-in VSE class hierarchy. Methods in each class are specified by using VSE's very high-level object-oriented scripting language. Vehicles are modeled as dynamic objects and are instantiated at run time with respect to the interarrival times sampled from the probability distributions shown in Table 10.1. Each graphical object in the model representation is set to belong to a class to inherit the characteristics and behavior specified in that class.

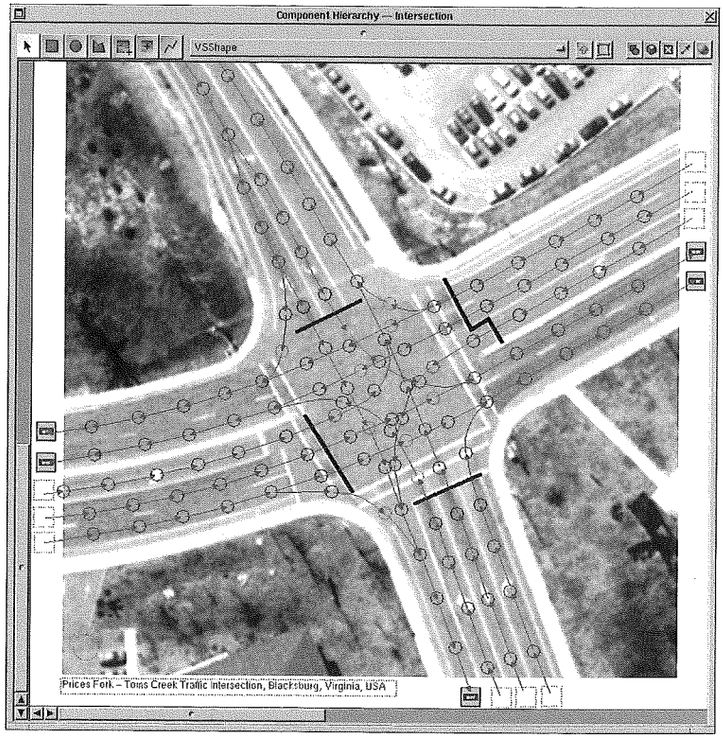

Figure 10.3 Model specification in the Visual Simulation Environment.

10.2.6 Programming

Translation of the communicative model (model specification) into a programmed model (executable model) constitutes the process of programming. A *programmed model* is an executable simulation model representation which does not incorporate an experiment design. The process of programming can be performed by the modeler using a simulation software product [19], a simulation programming language [19], or a high-level programming language [20].

Case Study. The traffic intersection model specification is created by using the VSE Editor tool. Then, by selecting the "Prepare for Simulation" menu option, the model specification is automatically translated into an executable form. The VSE Simulator tool is used to execute and animate the model and conduct experiments.

10.2.7 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 [10]. 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 that maximize or minimize the value of a response variable [11]. *Factorial designs* can be employed to determine the effect of various input variables on a response variable (Chapter 6; [21]). *Variance reduction techniques* can be implemented to obtain greater statistical accuracy for the same amount of simulation [11]. *Ranking and selection techniques* can be utilized for comparing alternative systems (Chapter 8; [11, 19]). Several methods (e.g., replication, batch means, regenerative) can be used for statistical analysis of simulation output data (Chapter 7).

Case Study. The VSE Simulator's design of experiments panel is used to specify the method of replications for statistical analysis of simulation output data. Fourteen performance measures are defined for all travel paths:

W_{jL} = average waiting time of vehicles arriving and turning left in lane j,
$j = 1, 3, 6, 9$

W_{jS} = average waiting time of vehicles arriving and travelling straight in lane j,
$j = 2, 4, 5, 7, 10, 11$

W_{jR} = average waiting time of vehicles arriving and turning right in lane j, $j = 2, 5, 8, 11$

The waiting time is the time spent by a vehicle in the entire traffic intersection, from arrival to the intersection to departure. The experimentation objective is to select the best traffic light timing out of three alternatives: the current light timing and two other alternatives suggested based on observation. The best light timing produces the lowest W_{jL}, W_{jS}, and W_{jR} for each lane j.

10.2.8 Experimentation

This is the process of experimenting with the simulation model for a specific purpose. Some purposes of experimentation are [10] (1) training, (2) comparison of different operating policies, (3) evaluation of system behavior, (4) sensitivity analysis, (5) forecasting, (6) optimization, and (7) determination of functional relations. The process of experimentation produces the simulation results.

Case Study. Using the VSE Editor, the model is instrumented to collect data on each performance measure. The model is warmed up for a total of 1000 vehicles passing through the intersection. Data are collected during the steady-state period of 10,000 vehicles. Identical experimental conditions are created by way of using the same random number generator seeds for each traffic light timing alternative. The model is replicated 30 times and each performance measure replication value is written to output file f, where f = 1, 2, 3, ... , 14. Then the VSE Output Analyzer tool is used to open the output files and construct confidence intervals and provide general statistics for each performance measure.

10.2.9 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.

Case Study. Using the VSE Editor, the traffic light timing is modified and a new executable model is produced. The VSE Simulator is used to conduct experiments with the model under the new traffic light timing. The VSE Output Analyzer is used to construct confidence intervals and provide general statistics for each performance measure.

10.2.10 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, deciding on 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.

Case Study. The experimentation results under three traffic light timing alternatives are presented to the decision makers. There was not a single alternative that reduced the average waiting time in every travel path. However, alternative 1 was found to reduce the waiting times to acceptable levels in all travel paths and hence it is accepted and implemented by the decision makers.

10.3 VERIFICATION, VALIDATION, AND TESTING PRINCIPLES

According to Webster's Encyclopedic Unabridged Dictionary, a *principle* is defined as "1. an accepted or professed rule of action or conduct. 2. a fundamental, primary, or general law or truth from which others are derived. 3. a fundamental doctrine or tenet; a distinctive ruling opinion." All three definitions above apply to the way the term *principle* is used herein.

Principles are important to an understanding of the foundations of VV&T. The principles help researchers, practitioners, and managers better comprehend what VV&T is all about. They serve to provide the underpinnings for over 75 VV&T techniques, described in Section 10.4, that can be used throughout the life cycle. Understanding and applying these principles is crucially important for the success of a simulation study.

The 15 principles presented herein are established based on the experience described in the published literature and the author's experience during his VV&T research since 1978. The principles are listed below in no particular order.

Principle 1 VV&T must be conducted throughout the entire life cycle of a simulation study.

VV&T is not a phase or step in the life cycle but a continuous activity throughout the entire life cycle presented in Figure 10.1. Conducting VV&T for the first time in the life cycle when the experimental model is complete is analogous to a teacher who gives only a final examination [22]. 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 students about their deficiencies so that they can study more to improve their knowledge as the course progresses.

The situation in 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 problem definition to the presentation of simulation results. This allows us to identify and rectify quality deficiencies during the life-cycle phase in which they occur.

A simulation model goes through five levels of testing during its life cycle:

- *Level 1: Private Testing.* Performed by the modeler in private with no documentation. Although informal, private testing is strongly encouraged prior to formal submodel/module testing [23].
- *Level 2: Submodel (Module) Testing.* Planned, performed, and documented independently by the SQA group. Submodel testing treats each submodel as a stand-

alone unit, with its own input and output variables, that can be tested without other submodels.

- *Level 3: Integration Testing.* Planned, performed, and documented independently by the SQA group. Its objective is to substantiate that no inconsistencies in interfaces and communications between the submodels exist when the submodels are combined to form the model. It is assumed that each submodel has passed the submodel testing prior to integration testing.
- *Level 4: Model (Product) Testing.* Planned, performed, and documented independently by the SQA group. Its objective is to assess the validity of the overall model behavior.
- *Level 5: Acceptance Testing.* Planned, performed, and documented independently by the sponsor of the simulation study or the independent third party hired by the sponsor. Its objective is to establish the sufficient credibility of the simulation model so that its results can be accpeted and used by the sponsor.

Principle 2 The outcome of simulation model VV&T should not be considered as a binary variable where the model is absolutely correct or absolutely incorrect.

Since a model is an abstraction of a system, perfect representation is never expected. Shannon [10] indicates that "it is not at all certain that it is ever theoretically possible to establish if we have an absolutely valid model; even if we could, few managers would be willing to pay the price." The outcome of model VV&T should be considered as a degree of credibility on a scale from 0 to 100, where 0 represents absolutely incorrect and 100 represents absolutely correct. As depicted in Figure 10.4 [10, 24], as the degree of model credibility increases, so will the model development cost. At the same time, the model utility will also increase, but probably at a decreasing rate. The point of intersection of two curves changes from one model to another.

Principle 3 A simulation model is built with respect to the study objectives and its credibility is judged with respect to those objectives.

The objectives of a simulation study are identified in the formulated problem phase and explicitly and clearly specified in the system and objectives definition phase of the

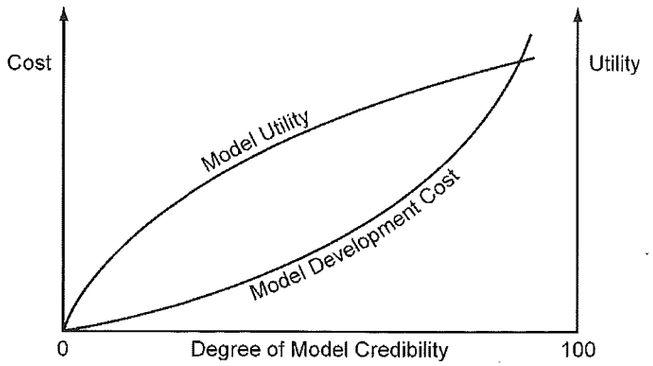

Figure 10.4 Model credibility versus cost and utility.

life cycle shown in Figure 10.1. Accurate specification of the study objectives is crucial for the success of a simulation study. The model is either developed from scratch or an existing model is modified for use or an available one is selected for use as is, *all* with respect to the study objectives.

The study objectives dictate how representative the model should be. Sometimes, 60% representation accuracy may be sufficient; sometimes, 95% accuracy may be required, depending on the importance of the decisions that will be made based on the simulation results. Therefore, model credibility must be judged with respect to the study objectives. The adjective *sufficient* must be used in front of the terms such as *model credibility*, *model validity*, or *model accuracy* to indicate that the judgment is made with respect to the study objectives. It is more appropriate to say that "the model is sufficiently valid" than to say that 'the model is valid." Here "sufficiently valid" implies that the validity is judged with respect to the study objectives and found to be sufficient.

Principle 4 Simulation model VV&T requires independence to prevent developer's bias.

Model testing is meaningful when conducted in an independent manner by an unbiased person. The model developer with the most knowledge of the model may be the least independent when it comes to testing. The developers are often biased because they fear that negative testing results can damage the credibility of the organization and may lead to the loss of future contracts.

The independence in model testing can be achieved in two ways: (1) establishing an SQA group within the organization conducting the simulation study, and (2) using an independent third party hired by the sponsor of the simulation study. The first one is required to achieve independence in level 3 and 4 testing within the organization as described under principle 1. The SQA group must be independent from the department in charge of conducting the simulation study and should report to the top management. The SQA group is responsible for planning, performing, and documenting all level 2, 3, and 4 tests in an unbiased manner. It should be made clear to the simulation project team that the main thrust of testing is to detect and document faults; it is *not* performance appraisal of the project team. This point must be communicated persuasively to everyone involved so that full cooperation is achieved in discovering and documenting errors.

The second one is required to achieve independence in level 5 (acceptance) testing as described under principle 1. The requirements for acceptance testing must be specified in the legal contract by the sponsor. The test cases to be used must be well documented. Although the sponsor can perform the acceptance testing, it is recommended that an independent third party contracted by the sponsor be responsible for the testing. In that way, the organization conducting the simulation study cannot claim that the sponsor is biased.

Principle 5 Simulation model VV&T is difficult and requires creativity and insight.

One must thoroughly understand the entire simulation model so as to design and implement effective tests and identify adequate test cases. Knowledge of the problem domain, expertise in the modeling methodology and prior modeling, and VV&T experience are required for successful testing. However, it is not possible for one person to fully understand all aspects of a large and complex model, especially if the model is a stochastic one containing hundreds of concurrent activities. The fundamental human limitation, called the *Hrair limit*, indicates that a human being cannot process more

than 7 ± 2 entities simultaneously. Hence testing a complex simulation model is a very difficult task that requires creativity and insight.

A model's developers are usually the most qualified to show the creativity and insight required for successful testing since they are intimately knowledgeable about the internals of a model. However, they are usually biased when it comes to model testing and they cannot be fully utilized. Therefore, the inability to use model developers effectively for testing increases the difficulty of testing. False beliefs exist about testing, as indicated by Hetzel [22]: "Testing is easy; anyone can do testing; no training or prior experience is required." The difficulty of model VV&T must not be underestimated. The model testing must be well planned and administered by the SQA group.

Principle 6 Simulation model credibility can be claimed only for the conditions for which the model is tested.

The accuracy of the input–output transformation of a simulation model is affected by the characteristics of the input conditions. The transformation that works for one set of input conditions may produce absurd output when conducted under another set of input conditions. In the case study, for example, a stationary simulation model is built assuming constant arrival rate of vehicles during the evening rush hour, and its credibility may be judged sufficient with respect to the evening rush-hour input conditions. However, the simulation model will show invalid behavior when run under the input conditions of the same traffic intersection between 7:00 A.M. and 6:00 P.M. During this time period, the arrival rate of vehicles is not constant and a nonstationary simulation model is required. Hence establishing sufficient model credibility for the evening rush-hour conditions does not imply sufficient model credibility for input conditions during other times. The prescribed conditions for which the model credibility has been established is called the *domain of applicability* of the simulation model [25]. Model credibility can be claimed only for the domain of applicability of the model.

Principle 7 Complete simulation model testing is not possible.

Exhaustive (complete) testing requires that the model is tested under *all* possible input conditions. Combinations of feasible values of model input variables can generate millions of logical paths in the model execution. Due to time and budgetary constraints, it is impossible to test the accuracy of millions of logical paths. Therefore, in model testing, the purpose is to increase our confidence in model credibility as much as dictated by the study objectives rather than trying to test the model completely. How much to test or when to stop testing is dependent on the desired domain of applicability of the simulation model. The larger the domain, the more testing is required. The domain of applicability is determined with respect to the study objectives.

Hundreds of logical paths may need to be tested so as to substantiate model credibility under a set of prescribed conditions. Due to budgetary and time constraints, all logical paths may not be tested. Test data or test cases are prepared to test the logical paths in a random manner. Test data can be generated by using (1) random values, (2) deterministic values, (3) minimum values for all input variables, (4) maximum values for all input variables, (5) a mixture of minimum and maximum values for all input variables, (6) invalid values, and (7) simulated values.

When using test data, it must be noted that the law of large numbers simply does not apply. The question is not how much test data is used, but what percentage of the valid input domain is covered by the test data. The higher the percentage of coverage, the higher the confidence we can gain in model credibility.

Principle 8 Simulation model VV&T must be planned and documented.

Testing is not a phase or step in the model development life cycle; it is a continuous activity *throughout* the entire life cycle. The tests should be identified, test data or cases should be prepared, tests should be scheduled, and the entire testing process should be documented. Ad hoc or haphazard testing does not provide reasonable measurement of model accuracy. Hetzel [22] points out that "such testing may even be harmful in leading us to a false sense of security." Careful planning is required for successful testing.

Planning and documenting model testing involves at least three groups of people: (1) sponsor of the simulation study, (2) SQA group of the organization conducing the simulation study, and (3) simulation project management. The sponsor is responsible for documenting the tests and specifying the test cases or data with which the acceptance testing will be performed. It is recommended that a plan for acceptance testing be made part of the legal contract between sponsor and contractor of the simulation study. If the study is conducted internally within an organization, acceptance testing plan should be part of the requirements specification document. The SQA group is responsible for planning, performing, and documenting level 2 (module), level 3 (integration), and level 4 (model) testing.

A *test plan* is a document describing what is selected for testing, test database and code, test specifications, standards and conventions, test control, test configuration, test tools, and the results expected. An acceptance test plan is presented by Beizer [23].

Principle 9 Type I, II, and III errors must be prevented.

Three types of errors may be committed in conducting a simulation study as depicted in Figure 10.5 [9]. A *type I error* is committed when the simulation results are rejected when in fact they are sufficiently credible. A *type II error* occurs when invalid simulation results are accepted as if they are sufficiently valid. A *type III error* occurs if the wrong problem is solved and committed when the problem formulated does not completely contain the actual problem.

Committing a type I error unnecessarily increases the cost of model development. The consequences of type II and type III errors can be catastrophic, especially when critical decisions are made on the basis of simulation results. A type III error implies that the problem solution and the simulation study results will be irrelevant when it is committed.

The probability of committing a type I error is called *model builder's risk* and the probability of committing a type II error is called *model user's risk* [26]. VV&T activities must focus on minimizing these risks as much as possible. Balci and Sargent [26] show how to quantify these risks when using hypothesis testing for the validation of a simulation model with two or more output variables.

Figure 10.5 illustrates the occurrence of three types of errors, assuming that the simulation study results are certified by an independent organization. Whenever feasible, simulation results should be independently certified so as to remove the developer's bias and promote *independent* VV&T (see Principle 4).

Principle 10 Errors should be detected as early as possible in the life cycle of a simulation study.

A rush to model implementation is a common problem in simulation studies. Sometimes simulation models are built by direct implementation in a simulation system or (simulation) programming language with no or very little formal model specification. As a result of this harmful build-and-fix approach, experimental model VV&T becomes the only main credibility assessment stage.

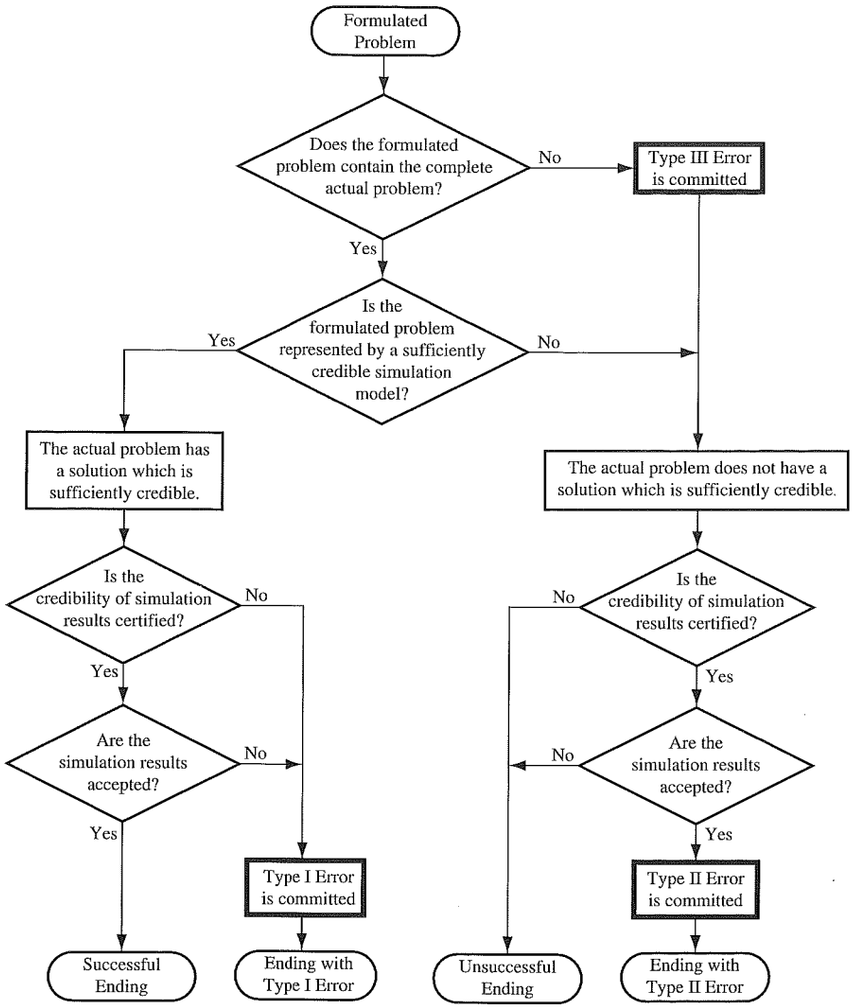

Figure 10.5 Type I, II, and III errors in a simulation study.

Detection and correction of errors as early as possible in the life cycle of a simulation study must be the primary objective. Sufficient time and energy must be expended for each VV&T activity shown in Figure 10.1. Nance [5] points out that detecting and correcting major modeling errors during the process of model implementation and in later phases is very time consuming, complex, and expensive. Some vital errors may not be detectable in later phases, resulting in the occurrence of a type II or III error.

Nance and Overstreet [27] advocate this principle and provide diagnostic testing techniques for models represented in the form of condition specification. A model analyzer software tool is included in the definition of a simulation model development environment so as to provide effective early detection of model specification errors [16, 28, 29].

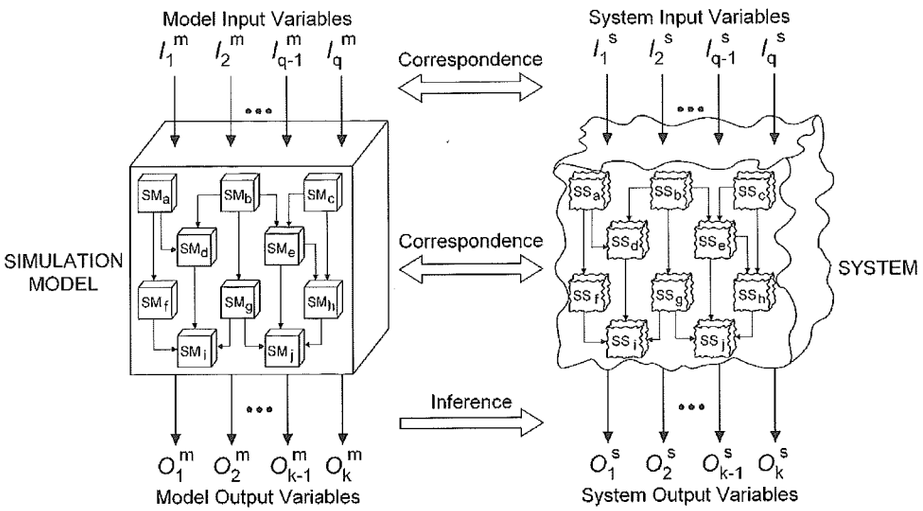

Figure 10.6 Model and system characteristics.

Principle 11 Multiple response problem must be recognized and resolved properly.

Figure 10.6 shows a simulation model with k output variables (responses or performance measures) and q input variables representing a system with corresponding k output variables and q input variables. Superscript m indicates model and s indicates system. SM stands for submodel and SS stands for subsystem.

Due to the multiple response problem described by Shannon [10], the validity of a simulation model with two or more output variables (responses) cannot be tested by comparing the corresponding model and system output variables one at a time (i.e., O_1^m versus O_1^s, O_2^m versus O_2^s, ..., O_k^m versus O_k^s, as shown in Figure 10.6) using a univariate statistical procedure. A multivariate statistical procedure must be used to incorporate the correlations among the output variables in the comparison. Two such multivariate statistical techniques are presented by Balci and Sargent [30] using Hotelling's T^2-statistic for constructing ellipsoidal joint confidence regions to assess model validity. The techniques are described below.

The first technique requires independence between the model and system output data and is intended for self-driven simulation models of observable systems. Assume that the model and system each has k output variables, as depicted in Figure 10.6, with n observations on each model output variable and N observations on each system output variable. Let $(\boldsymbol{\mu}^m)' = [\mu_1^m, \mu_2^m, \ldots, \mu_k^m]$ and $(\boldsymbol{\mu}^s)' = [\mu_1^s, \mu_2^s, \ldots, \mu_k^s]$ be the k-dimensional vectors of population means of model and system output variables, respectively. Let $(\bar{\mathbf{o}}^m)' = [\bar{o}_1^m, \bar{o}_2^m, \ldots, \bar{o}_k^m]$ and $(\bar{\mathbf{o}}^s)' = [\bar{o}_1^s, \bar{o}_2^s, \ldots, \bar{o}_k^s]$ be the k-dimensional vectors of sample means of observations on model and system output variables, respectively. Then, the $100(1-\gamma)\%$ joint confidence region is specified by those vectors $\boldsymbol{\delta} = \boldsymbol{\mu}^m - \boldsymbol{\mu}^s$ satisfying the inequality

$$(\bar{\mathbf{o}}^m - \bar{\mathbf{o}}^s - \boldsymbol{\delta})' S^{-1} (\bar{\mathbf{o}}^m - \bar{\mathbf{s}}^s - \boldsymbol{\delta}) \leq \frac{n+N}{nN} T^2_{\gamma;k,n+N-k-1} \tag{1}$$

where S is the pooled variance–covariance matrix and $T^2_{\gamma;k,n+N-k-1}$ is the upper γ percentage point of Hotelling's T^2-distribution with degrees of freedom k and $n+N-k-1$.

The second technique requires paired observations between the model and system output variables and is intended for trace-driven simulation models. Let $\bar{\mathbf{d}}' = [\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_k]$ be the k-dimensional vector of sample means of differences between the paired observations on the model and system output variables with a sample size of N. The $100(1-\gamma)\%$ joint confidence region consists of the vectors $\boldsymbol{\mu}^d$ satisfying the inequality

$$N(\bar{\mathbf{d}} - \boldsymbol{\mu}^d)' S_d^{-1} (\bar{\mathbf{d}} - \boldsymbol{\mu}^d) \leq T^2_{\gamma;k,N-k} \tag{2}$$

where S_d is the variance–covariance matrix of the differences.

When $k = 3$, the joint confidence region can be presented visually as illustrated in Figure 10.7 and can be used to assess the model accuracy with an exact level of $100(1-\gamma)\%$ confidence. We can conclude that we are $100(1-\gamma)\%$ confident that the differences between the population means of corresponding model and system output variables are contained within the joint confidence region shown in Figure 10.7. Ideally, the joint confidence region contains zero at its center, and the smaller its size, the better it is. Any deviation from the idealistic case is an indication of the degree of invalidity. As k increases, interpretation of the joint confidence region becomes difficult but

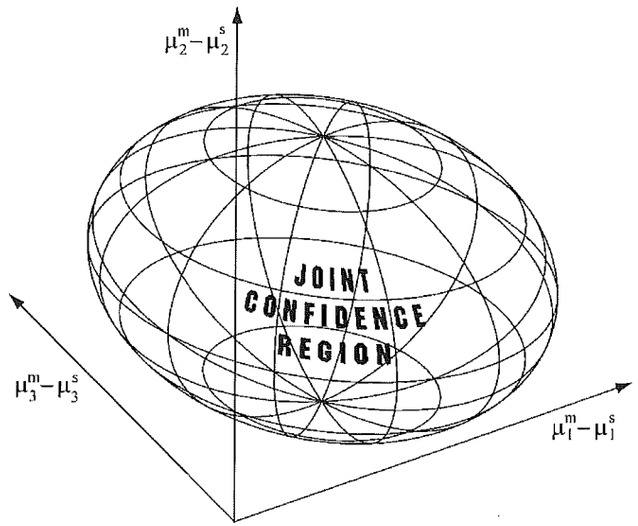

Figure 10.7 Joint confidence region representing the validity of a simulation model with three output variables.

not impossible. With the use of computer-aided assistance, the model validity can be examined.

Principle 12 Successfully testing each submodel (module) does not imply overall model credibility.

Suppose that a simulation model is composed of submodels (SM_x) representing subsystems (SS_x) respectively, as depicted in Figure 10.6. Submodel x can be tested individually by comparison to subsystem x, where $x = a, b, \ldots, j$, using many of the VV&T techniques described in Section 10.4. The credibility of each submodel is judged to be sufficient with some error that is acceptable with respect to the study objectives. We may find each submodel to be sufficiently credible, but this does not imply that the whole model is sufficiently credible. The allowable errors for the submodels may accumulate to be unacceptable for the entire model. Therefore, the entire model must be tested even if each submodel is found to be sufficiently credible.

Principle 13 Double validation problem must be recognized and resolved properly.

If data can be collected on both system input and output, model validation can be conducted by comparing model and system outputs obtained by running the model with the "same" input data that drives the system. Determination of the "same" is yet another validation problem within model validation. Therefore, this is called the *double validation problem.* This is an important problem that is often overlooked. It greatly affects the accuracy of model validation. If invalid input data models are used, we may still find the model and system outputs sufficiently matching each other and conclude incorrectly on the sufficient validity of the model.

The "same" is determined by validating the input data models. In the case study, the input data models are the probability distributions given in Table 10.1. We must substantiate that the input data models have sufficient accuracy in representing the system input process. Input data modeling deals with characterization of the system input data [13, 31]. Simulation models are categorized into two classes with respect to the way they are driven: trace-driven and self-driven. In *trace-driven simulation*, the system input is characterized by the trace data collected from the instrumented system. The trace data become the input data model which should be validated against the actual system input process.

In *self-driven simulations*, the simulation model is driven by randomly sampling from the probabilistic models developed to represent the data collected on the system input process. Usually, input data modeling is achieved by fitting standard probability distributions to observed data. The input data models should be constructed using a multivariate statistical approach if the input variables are correlated. Individually building a probabilistic model for each input variable does not incorporate the correlations among the input variables; therefore, a multivariate probabilistic approach should be used.

Principle 14 Simulation model validity does not guarantee the credibility and acceptability of simulation results.

Model validity is a necessary but not a sufficient condition for the credibility and acceptability of simulation results. We assess model validity with respect to the study objectives by comparing the model with the system as it is defined. If the study objectives are identified incorrectly and/or the system is defined improperly, the simulation results will be invalid; however, we may still find the model to be sufficiently valid

by comparing it with the improperly defined system and with respect to the incorrectly identified objectives.

A distinct difference exists between the model credibility and the credibility of simulation results. Model credibility is judged with respect to the system (requirements) definition and the study objectives, whereas the credibility of simulation results is judged with respect to the actual problem definition and involves the assessment of system definition and identification of study objectives. Therefore, model credibility assessment is a subset of credibility assessment of simulation results.

Principle 15 Formulated problem accuracy greatly affects the acceptability and credibility of simulation results.

It has been said that a problem correctly formulated is half solved [32]. Albert Einstein once indicated that the correct formulation of a problem was even more crucial than its solution. The ultimate goal of a simulation study should not be just to produce a solution to a problem but to provide one that is sufficiently credible and accepted and implemented by the decision makers. We cannot claim that we conducted an excellent simulation study but the decision makers did not accept our results and we cannot do anything about it. Ultimately we are responsible for the acceptability and usability of our simulation solutions, although in some cases we cannot affect or control the acceptability.

Formulated problem accuracy assessed by conducting formulated problem VV&T greatly affects the credibility and acceptability of simulation results. Insufficient problem definition and inadequate sponsor involvement in defining the problem are identified as two important problems in the management of computer-based models. It must be recognized that if problem formulation is poorly conducted, resulting in an incorrect problem definition, no matter how fantastically we solve the problem, the simulation study results will be irrelevant. Balci and Nance [9] present an approach to problem formulation and 38 indicators for assessing the formulated problem accuracy.

10.4 VERIFICATION, VALIDATION, AND TESTING TECHNIQUES

More than 75 VV&T techniques are presented in this section. Most of these techniques come from the software engineering discipline and the remaining are specific to the modeling and simulation field. The software VV&T techniques selected which are applicable for simulation model VV&T are presented in a terminology understandable by a simulationist. Descriptions of some software VV&T techniques are changed so as to make them directly applicable and understandable for simulation model VV&T.

Figure 10.8 shows a taxonomy that classifies the VV&T techniques into four primary categories: informal, static, dynamic, and formal. A primary category is further divided into secondary categories, shown in italics. The use of mathematical and logic formalism by the techniques in each primary category increases from informal to formal from left to right. Similarly, the complexity also increases as the primary category becomes more formal.

It should be noted that some of the categories presented in Figure 10.8 possess similar characteristics and in fact have techniques that overlap from one category to another. However, a distinct difference between each classification exists, as it is evident in the discussion of each in this section. The categories and techniques in each category are described on page 355.

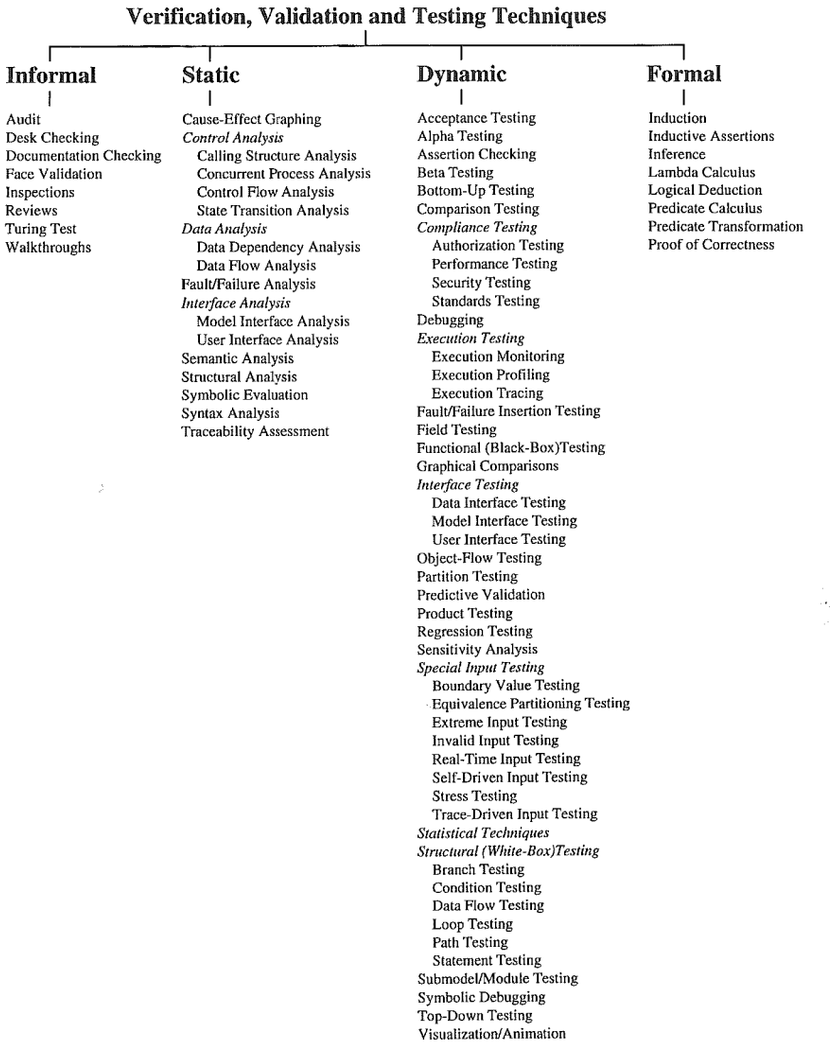

Figure 10.8 Taxonomy of verification, validation, and testing techniques.

10.4.1 Informal VV&T Techniques

Informal techniques are among the most commonly used. 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 a lack of structure or formal guidelines for the use of the techniques. In fact, these techniques are applied using well-structured approaches under formal guidelines and they can be very effective if employed properly.

Audit. An audit is undertaken to assess how adequately the simulation study is conducted with respect to established plans, policies, procedures, 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. The process of documenting and retaining sufficient evidence about how the accuracy is substantiated is called an *audit trail* [33]. Auditing is carried out on a periodic basis through a mixture of meetings, observations, and examinations [34]. Audit is a staff function and serves as the "eyes and ears of management" [33].

Desk Checking. *Desk checking* (also known as *self-inspection*) 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 [35]. Syntax checking, cross-reference checking, convention violation checking, detailed comparison to specification, reading the code, the control flow graph analysis, and path sensitizing should all be conducted as part of desk checking [23].

Documentation Checking. Documentation checking is conducted to ensure correctness, completeness, consistency, and unambiguity of all model documentation and to justify that all documentation is up to date with respect to model logic specification. Often, a model component logic is modified but the component's documentation is not updated. Sometimes model logic is documented erroneously. In the case study, the documentation delivered to the decision makers must be an accurate and up-to-date description of model logic and its results.

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 under identical input conditions and judge whether the model and its results are reasonable. Face validation is useful as a preliminary approach to validation [36]. In the case study, the confidence intervals for the 14 performance measures obtained under the currently used traffic light timing can be presented to experts. The experts can judge if average waiting times of vehicles are reasonable under the rush-hour traffic conditions observed.

Inspections. Inspections are conducted by a team of four to six members for any model development phase such as system and objectives definition, conceptual model design, or communicative model design. For example, in the case of communicative model 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:* representative of the team that created the model design
5. *Implementer:* translates the model design into code (programmed model).
6. *Tester:* SQA group representative.

An inspection goes through five distinct phases: overview, preparation, inspection, rework, and follow-up [37]. 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 for 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.

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 [23, 33, 37–40].

Reviews. The review is conducted in a manner similar to that of 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 model development process is being conducted according to stated study objectives and to 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, (6) model structuredness, (7) model consistency, (8) model completeness, and (9) 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.

In contrast with 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 toward the early stages of the model development life cycle [33, 34, 41, 42].

Turing Test. The Turing test is based on the expert knowledge of people about the system under study. The experts 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 experts 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 [44–45].

In the case study, two confidence intervals are first constructed for the average waiting time of vehicles arriving and turning left in lane 1: (1) confidence interval estimated via simulation, and (2) confidence interval constructed based on data collected at the traffic intersection. The two sets of confidence intervals are presented to an expert who has intimate knowledge of the traffic intersection operation. Without identifying which one is which, the expert is asked to differentiate between the two. If the expert cannot identify which one belongs to the real traffic intersection, the model is considered sufficiently valid with respect to that performance measure. This process is repeated for each of the 14 performance measures.

Walkthroughs. Walkthroughs are conducted by a team composed of a coordinator, model developer, and three to six other members. All members other than the model developer should not be directly involved in the development effort. A typical structured walkthrough team consists of:

1. *Coordinator:* most often the SQA group representative who organizes, moderates, and follows up the walkthrough activities
2. *Presenter:* most often 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
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 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 [35, 46–49].

10.4.2 Static VV&T Techniques

Static VV&T techniques are concerned with accuracy assessment on the basis of characteristics of the static model design and source code. Static techniques do not require machine execution of the model, but mental execution can be used. The techniques are very popular and widely used, with many automated tools available to assist in the VV&T process. 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, modeling techniques and practices employed, data and control flow within the model, and syntactical accuracy [42].

Cause–Effect Graphing. Cause–effect graphing assists model correctness assessment 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 reflected accurately in the model specification. In the case study 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 [42, 48, 50].

Control Analysis. The control analysis category consists of the following techniques that are used for the analysis of the control characteristics of the model:

1. *Calling Structure Analysis:* used to assess model accuracy by identifying who *calls* whom and who is *called* by whom. The "who" could be a module, procedure, subroutine, function, or a method in an object-oriented model [51]. In the case study, inaccuracies caused by message passing (e.g., sending a message to a nonexistent object) in the object-oriented traffic intersection VSE model can be revealed by analyzing which methods invoke a method and by which methods a method is invoked.
2. *Concurrent Process Analysis:* especially useful for parallel and distributed simulations presented in Chapter 13. Model accuracy is assessed by analyzing the overlap or concurrency of model components executed in parallel or as distributed. Such analysis can reveal synchronization problems such as deadlocks [52].
3. *Control Flow Analysis:* requires the development of a graph of the model where conditional branches and model junctions are represented by nodes and the model segments between such nodes are represented by links [23]. A node of the model graph usually represents a logical junction where the flow of control changes, while an edge represents toward 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 [27] propose several diagnostics based on analysis of graphs constructed from a particular form of model specification called *condition specification* [14, 53]. The diagnostic assistance is categorized into three parts:

1. *Analytical:* determination of the existence of a property
2. *Comparative:* measures of differences among multiple model representations
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 classiciation, precedence structure, decomposition, and run-time graph [27].

4. *State Transition Analysis:* requires the identification of a finite number of states the model execution goes through. A state transition diagram is created showing how the model transitions from one state to another. Model accuracy is assessed by analyzing the conditions under which a state change occurs. This technique is especially effective for those simulation models created under the activity scanning, three-phase, and process interaction conceptual frameworks [20].

Data Analysis. The data analysis category consists of several techniques that are used to ensure that (1) proper operations are applied to data objects (e.g., data structures, event lists, linked lists), (2) the data used by the model are properly defined, and (3) the defined data are properly used [33]:

1. *Data Dependency Analysis:* involves the determination of what variables depend on what other variables [54]. For parallel and distributed simulations, the data dependency knowledge is critical for assessing the accuracy of process synchronization.

2. *Data Flow Analysis:* 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 [35] (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 [42, 55].

Fault/Failure Analysis. Fault (incorrect model component)/failure (incorrect behavior of a model component) analysis uses model input–output transformation descriptions to identify how the model *might* logically fail. The model design specification is examined to determine if any failure-mode possibilities could logically occur and in what context and under what conditions. Such model examinations often lead to identification of model defects [51].

Interface Analysis. The interface analysis category consists of several techniques that are especially useful for verification and validation of interactive and distributed simulations:

1. *Model Interface Analysis:* conducted to examine the (sub)model-to-(sub)model interface and determine if the interface structure and behavior are sufficiently accurate.

2. *User Interface Analysis:* conducted to examine the user–model interface and determine if it is human engineered so as to prevent occurrences of errors during the user's interactions with the model. It is also used to assess how accurately the interface is integrated with the simulation model. This technique is particularly useful for accuracy assessment of interactive simulation models used for training purposes.

Semantic Analysis. Semantic analysis is conducted by the simulation system translator or a 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 [42]:

1. *Symbol Tables:* the elements or symbols that are manipulated in the model, function declarations, type and variable declarations, scoping relationships, interfaces, dependencies, and so on.
2. *Cross-Reference Tables:* 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:* describe the actual interfaces of the caller and the called.
4. *Maps:* relate the generated run-time code to the original source code.
5. *"Pretty Printers" or Source Code Formatters:* provide reformatted source listing on the basis of its syntax and semantics, clean pagination, highlighting of data elements, and marking of nested control structures.

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 [56, 57] apply the theory of computational complexity and show that the problem of verifying structural properties of simulation models is 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.

Symbolic Evaluation. Symbolic evaluation is used to assess model accuracy by exercising 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 = -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 as to 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. Eventually, all paths are taken.

The result of the execution can be represented graphically as a symbolic execution tree [35, 58]. 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 evaluation 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 [58–60].

Syntax Analysis. Syntax analysis is carried on by the simulation software compiler or simulation programming language compiler to assure that the mechanics of the language are applied correctly [23]. In the case study, during model preparation, the VSE Editor lists all syntax errors in the preparation window as shown in Figure 10.9. Double-clicking a syntax error name displays the method where the error occurs, draws a rectangle around the statement, and highlights the token as a potential source of error.

Traceability Assessment. The traceability assessment is used to match, one to one, the elements of one form of the model to another. For example, the elements of the system and objectives definition (requirements specification) are matched one to one to the elements of the communicative model (design specification). Unmatched elements *may* reveal either unfulfilled requirements or unintended design functions [51].

10.4.3 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 or stubs) 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.

Dynamic VV&T techniques are usually applied using the following three steps. In step 1 the executable 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. For example, consider the worldwide air traffic control and satellite communication object-oriented visual simulation model created by using the VSE [16–18] in

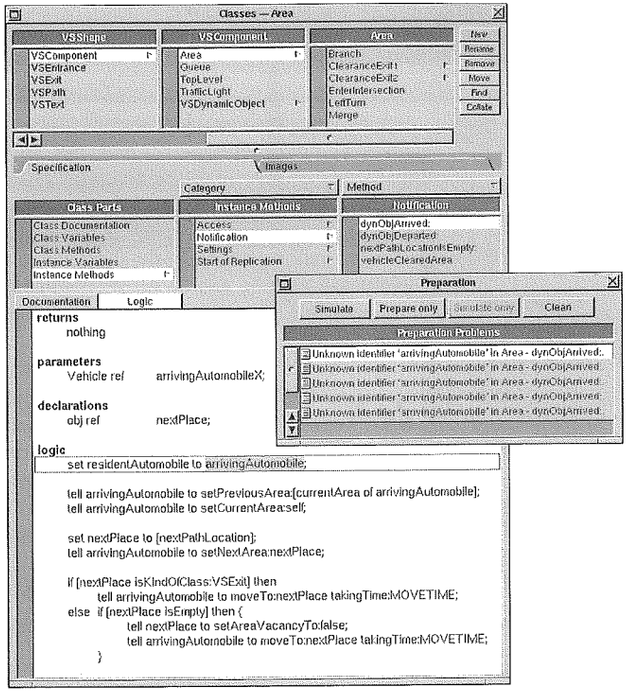

Figure 10.9 Identification of syntax errors during model preparation.

Figure 10.10. The model can be instrumented in step 1 to record the following information every time an aircraft enters the coverage area of a satellite: (1) aircraft tail number; (2) current simulation time; (3) aircraft's longitude, latitude, and altitude; and (4) satellite's position and identification number. In step 2 the model is executed and the information collected is written to an output file. In step the output file is examined to reveal discrepancies and inaccuracies in model representation.

Acceptance Testing. Acceptance testing is conducted either by the client organization, by the developer's SQA group in the presence of client representatives, or by an independent contractor hired by the client after the model is officially delivered and before the client officially accepts the delivery. The model is operationally tested by using the actual hardware and actual data to determine whether all requirements specified in the legal contract are satisfied [33, 37].

Alpha Testing. Alpha testing refers to the operational testing of the alpha version of the complete model at an inhouse site that is not involved with the model development [23].

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

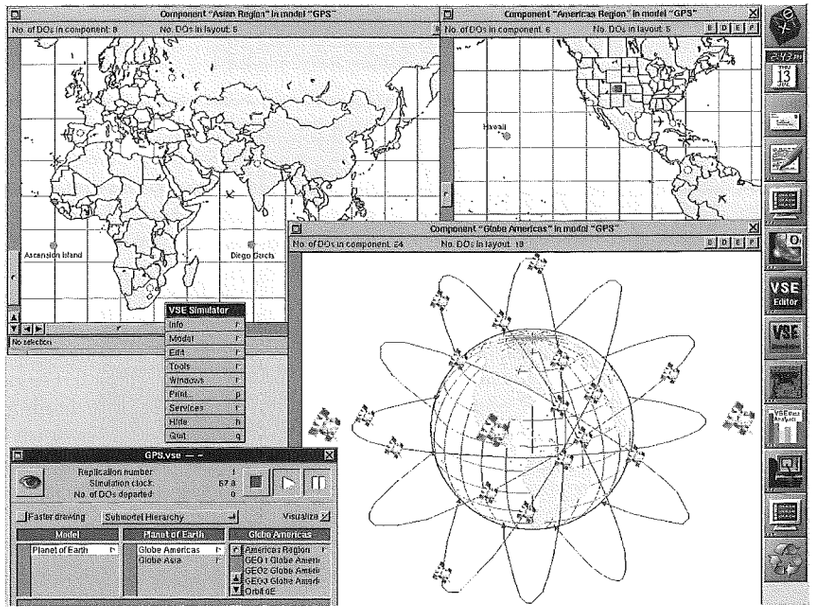

Figure 10.10 Visual simulation of global air traffic control and satellite communication. (From ref. 16.)

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 entire 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 [61]:

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

The "optional-qualifiers" may be chosen such as `all, some, after jth aircraft, before time t`. The "control options" may have the following example syntax [61]:

$$\ldots \text{ LIMIT n [VIOLATIONS]} \left[\left\{ \begin{matrix} \text{HALT} \\ \text{EXIT [VIA] procedure} - \text{name} \end{matrix} \right\} \right] \ldots$$

Consider, for example, the following pseudocode [42]:

```
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 nonnegative. 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);
```

Assertion checking is also used to prevent structural model inaccuracies. For example, the model in Figure 10.10 can contain assertions such as (1) a satellite communicates with the correct ground station, (2) an aircraft's tail number matches its type, and (3) an aircraft's flight path is consistent with the official airline guide.

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 forcing the modeler to make a trade-off between execution efficiency and accuracy. If the execution performance is critical, the assertions should be turned off but kept permanently to provide both documentation and means for maintenance testing [35].

Beta Testing. Beta testing refers to the operational testing of the beta version of the complete model as a "beta" user site under realistic field conditions [51].

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., those 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 [41].

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 the 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.

Comparison Testing. Comparison testing (also known as *back-to-back testing*) may be used when more than one version of a simulation model representing the same system is available for testing [41, 50]. For example, different simulation models may have been

developed to simulate the same military combat aircraft by different organizations or different simulation models may have been developed to represent the U.S. economy by different economists. All versions of the simulation model built to represent exactly the same system are run with the same input data and the model outputs are compared. Differences in the outputs reveal problems with model accuracy.

Compliance Testing. The compliance type of testing is intended to test how accurately different levels of access authorization are provided, how closely and accurately dictated performance requirements are satisfied, how well the security requirements are met, and how properly the standards are followed. These techniques are particularly useful for testing the federation of distributed and interactive simulation models under the Defense Department's high-level architecture (HLA) and distributed interactive simulation (DIS) architecture [62].

1. *Authorization Testing:* used to test how accurately and properly different levels of access authorization are implemented in the simulation model and how properly they comply with the established rules and regulations. The test can be conducted by attempting to execute a classified model within a federation of distributed models or try to use classified input data for running a simulation model without proper authorization [33].
2. *Performance Testing:* used to test whether (1) all performance characteristics are measured and evaluated with sufficient accuracy, and (2) all established performance requirements are satisfied [33].
3. *Security Testing:* used to test whether all security procedures are implemented correctly and properly in conducting a simulation exercise. For example, the test can be conducted by attempting to penetrate the simulation exercise while it is ongoing and break into classified components such as secured databases. Security testing is applied to substantiate the accuracy and evaluate the adequacy of the protective procedures and countermeasures [33].
4. *Standards Testing:* used to substantiate that the simulation model is developed with respect to the required standards, procedures, and guidelines.

Debugging. Debugging is an iterative process whose purpose it 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 reexecuted 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 [63].

Execution Testing. The execution testing category consists of several techniques that are used to collect and analyze execution behavior data for the purpose of revealing model representation errors:

1. *Execution Monitoring:* used to reveal errors by examining low-level information about activities and events that take place during model execution. It requires the instrumentation of a simulation model for the purpose of gathering data to provide activity- or event-oriented information about the model's dynamic behavior. For example, the model in Figure 10.10 can be instrumented to monitor the arrivals and departures of aircrafts within a particular city and the results can be compared with respect to the official airline guide to judge model validity. The model can also be instrumented to provide other low-level information, such as number of late arrivals, average passenger waiting time at the airport, and average flight time between two locations.

2. *Execution Profiling:* used to reveal errors by examining high-level information (profiles) about activities and events that take place during model execution. It requires the instrumentation of an executable model for the purpose of gathering data to present profiles about the model's dynamic behavior. For example, the model in Figure 10.10 can be instrumented to produce the following profiles to assist in model VV&T: (1) a histogram of aircraft interdeparture times, (2) a histogram of arrival times, and (3) a histogram of passenger checkout times at an airport.
3. *Execution Tracing:* used to reveal errors by "watching" the line-by-line execution of a simulation model. It requires the instrumentation of an executable model for the purpose of tracing the model's line-by-line dynamic behavior. For example, the model in Figure 10.10 can be instrumented to record all aircraft arrival times at a particular airport. Then the trace data can be compared against the official airline guide to assess model validity. 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 [64, 65].

Fault/Failure Insertion Testing. This technique is used to insert a kind of fault (incorrect model component) or a kind of failure (incorrect behavior of a model component) into the model and observe whether the model produces the invalid behavior as expected. Unexplained behavior may reveal errors in model representation.

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 [10, 45].

Functional Testing. Functional testing (also known as *black-box testing*) is used to assess the accuracy of model input–output transformation. It is applied by beeding 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 the millions. Therefore, the objective of functional 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 1000 input values (test data) does not imply high confidence in model input–output transformation accuracy just because 1000 is a large number. Instead, the number 1000 should be com-

pared with the number of allowable input values to determine what percentage 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 [48, 66].

In the case study, confidence intervals are constructed for each of the 14 performance measures by using actual observations collected on the vehicle waiting times. Confidence intervals are also constructed by using the simulation output data under the currently used traffic light timing. The actual and simulated confidence intervals with a confidence level of 95% are plotted corresponding to each performance measure. Lack of or little overlap between the actual and simulated confidence intervals revealed invalidity.

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 [67–70].

Interface Testing. The interface testing (also known as *integration testing*) category consists of several techniques that are used to assess the accuracy of data use, (sub)model-to-(sub)model interface, and user–model interface:

1. *Data Interface Testing:* conducted to assess the accuracy of data inputted into the model or outputted from the model during execution. All data interfaces are examined to substantiate that all aspects of data input–output are correct. This form of testing is particularly important for those simulation models whose inputs are read from a database and/or the results of which are stored into a database for later analysis. The model's interface to the database is examined to ensure correct importing and exporting of data [51].

2. *Model Interface Testing:* conducted to detect model representation errors created as a result of (sub)model-to-(sub)model interface errors or invalid assumptions about the interfaces. It is assumed that each model component (submodel) or a model in distributed simulation is individually tested and found to be sufficiently accurate before model interface testing begins.

This form of testing deals with how well the (sub)models are integrated with each other and is particularly useful for object-oriented and distributed simulation models. Under the object-oriented paradigm (see Chapter 11), objects (1) are created with public and private interfaces, (2) interface with other objects through message passing, (3) are reused with their interfaces, and (4) inherit the interfaces and services of other objects.

Model interface testing deals with the accuracy assessment of each type of four interfaces identified by Sommerville [41]:

a. *Parameter Interfaces:* pass data or function references from one submodel to another.

b. *Shared Memory Interfaces:* enable submodels to share a block of memory where data are placed by one submodel and retrieved from there by other submodels.

c. *Procedural Interfaces:* used to implement the concept of encapsulation under the object-oriented paradigm. An object provides a set of services (procedures) that can be used by other objects and hides (encapsulates) how a service is provided to the outside world.

d. *Message-Passing Interfaces:* enable an object to request the service of another by way of message passing.

Sommerville [41] classifies interface errors into three categories:

a. *Interface Misuse:* occurs when a submodel calls another and uses its interface incorrectly. For submodels with parameter interfaces, a parameter being passed may be of the wrong type, may be passed in the wrong order, or the wrong number of parameters may be passed.

b. *Interface Misunderstanding:* occurs when submodel A calls submodel B without satisfying the underlying assumptions of submodel B's interface. For example, submodel A calls a binary search routine by passing an unordered list to be searched when in fact the binary algorithm assumes that the list is already sorted.

c. *Timing Errors:* occur in real-time, parallel, and distributed simulations that use a shared memory or a message-passing interface.

3. *User Interface Testing:* conducted to detect model representation errors created as a result of user-model interface errors or invalid assumptions about the interfaces. This form of testing is particularly important for testing human-in-the-loop, interactive, and training simulations. User interface testing deals with the assessment of the interactions between the user and the model. The user interface is examined from low-level ergonomic aspects to instrumentation and controls and from human factors to global considerations of usability and appropriateness for the purpose of identifying potential errors [37, 50, 51].

Object-Flow Testing. Object-flow testing is similar to *transaction-flow testing* [23] and *thread testing* [41]. It is used to assess model accuracy by way of exploring the life cycle of an object during model execution. For example, a dynamic object (aircraft) can be marked for testing in the VSE model shown in Figure 10.10. Every time the dynamic object enters a model component, the visualization of that component is displayed. Every time the dynamic object interacts with another object within the component, the interaction is highlighted. Examination of how a dynamic object flows through the activities and processes and interacts with its environment during its lifetime in model execution is extremely useful for identifying errors in model behavior.

Partition Testing. Partition testing 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 that 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 that 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 [71]. 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 [72] 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 testing is capable of providing more comprehensive test data coverage than other test data generation techniques.

Predictive Validation. Predictive validation requires past input and output data of the system being modeled. 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 [73].

Product Testing. Product testing is conducted by the development organization after all submodels are successfully integrated (as demonstrated by the interface testing) and before acceptance testing by the client. No contractor wants to be in a situation where the product (model) fails the acceptance test. Product testing serves as a means of getting prepared for the acceptance testing. As such, the SQA group must perform product testing and make sure that all requirements specified in the legal contract are satisfied before delivering the model to the client organization [41]. As dictated by principle 12, testing each submodel successfully does not imply overall model credibility. Interface testing and product testing must be performed to substantiate overall model credibility.

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.

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 on model behavior [10]. 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 [36, 45, 74, 75].

Special Input Testing. The special input testing category consists of the following techniques that are used to assess model accuracy by way of subjecting the model to a variety of inputs:

1. *Boundary Value Testing:* 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 traveling straight as 0.45. This probabilistic branching can be implemented by using a uniform random number generator that produces numbers in the range $0 \leq RN \leq 1$. Thus three equivalence classes are identified: $0 \leq RN \leq 0.2$, $0.2 < RN \leq 0.55$, and $0.55 < RN \leq 1$. 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 the equivalence classes [48]. In the example above, the following test cases are selected for the left turn: 0.0, $\pm$ 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 that will cause the model to produce values on the boundaries of output equivalence classes [48]. The underlying rationale for this technique as a whole is that the most error-prone test cases lie along the boundaries [76]. 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.

2. *Equivalence Partitioning Testing:* partitions the model input domain into equivalence classes in such a manner that a test of a representative value from a class is assumed to be a test of all values in that class [33, 41, 50, 51].

3. *Extreme Input Testing:* conducted by running/exercising the simulation model by using only minimum values, only maximum values, or arbitrary mixture of minimum and maximum values for the model input variables.

4. *Invalid Input Testing:* performed by running/exercising the simulation model under incorrect input data and cases to determine whether the model behaves as expected. Unexplained behavior may reveal model representation errors.

5. *Real-Time Input Testing:* particularly important for assessing the accuracy of simulation models built to represent embedded real-time systems. For example, different design strategies of a real-time software system to be developed to control the operations of the components of a manufacturing system can be studied by simulation modeling. The simulation model representing the software design can be tested by way of running it under real-time input data that can be collected from the existing manufacturing system. Using real-time input data collected from a real system is particularly important to represent the timing relationships and correlations between input data points.

6. *Self-Driven Input Testing:* conducted by running/exercising the simulation model under input data randomly sampled from probabilistic models representing random phenomena in a real or futuristic system. A probability distribution (e.g., exponential, gamma, Weibull) can be fit to collected data or triangular and beta probability distributions can be used in the absence of data to model random input conditions (Chapter 3; [11, 12]). Then, using random variate generation techniques, random values can be sampled from the probabilistic models to test the model validity under a set of observed or speculated random input conditions.

7. *Stress Testing:* 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 10.10 can be stress tested by increasing the number of flights between two locations to an extremely high value. Such increase in workload may create unexpected high congestion in the model. Under stress testing, the model may exhibit invalid behavior; however, such behavior should be as expected and meaningfully documented [48, 63].

8. *Trace-Driven Input Testing.* conducted by running/exercising the simulation

TABLE 10.3 Statistical Techniques Proposed for Validation

Analysis of variance	[77]
Confidence intervals/regions	[10,11,30]
Factor analysis	[67]
Hotelling's T^2-tests	[10,26,78–80]
Multivariate analysis of variance	[81]
Standard MANOVA	
Permutation methods	
Nonparametric ranking methods	
Nonparametric goodness-of-fit tests	[77,82]
Kolmogorov–Smirnov test	
Cramer–Von Mises test	
Chi-square test	
Nonparametric tests of means	[10]
Mann–Whitney–Wilcoxon test	
Analysis of paired observations	
Regression analysis	[67,83,84]
Theil's inequality coefficient	[85–87]
Time-series analysis	
Spectral analysis	[45,84,88–90]
Correlation analysis	[91]
Error analysis	[92,93]
t-Test	[10,94]

model under input trace data collected from a real system. For example, a computer system can be instrumented by using software and hardware monitors to collect data by tracing all system events. The raw trace data is then refined to produce the real input data for use in testing the simulation model of the computer system.

Statistical Techniques. Much research has been conducted in applying statistical techniques for model validation. Table 10.3 presents the statistical techniques proposed for model validation and lists related references. The statistical techniques listed in the table require that the system being modeled be 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. As dictated by principle 11, a comparison of model and system multiple outputs must be carried out by using a multivariate statistical technique to incorporate the correlations among the output variables. A recommended validation procedure based on the use of simultaneous confidence intervals is presented below.

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 (CI) for the differences between the means of the jth model

and system output variables. The simultaneous confidence intervals (SCIs) formed by these CIs are called the *model range of accuracy* (MRA) [30].

Assume that there are k output variables from the model and k output variables from the system, as shown in Figure 10.6. Let

$$(\boldsymbol{\mu}^m)' = [\mu_1^m, \mu_2^m, \ldots, \mu_k^m] \quad \text{and} \quad (\boldsymbol{\mu}^s)' = [\mu_1^s, \mu_2^s, \ldots, \mu_k^s]$$

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 SCI to express the MRA for the mean behavior.

In approach I, the MRA is determined by the $100(1-\gamma)\%$ SCI for $\boldsymbol{\mu}^m - \boldsymbol{\mu}^s$ as

$$[\boldsymbol{\delta} - \boldsymbol{\tau}] \tag{3}$$

where $\boldsymbol{\delta}' = [\delta_1, \delta_2, \ldots, \delta_k]$ representing lower bounds and $\boldsymbol{\tau}' = [\tau_1, \tau_2, \ldots, \tau_k]$ representing upper bounds of the SCI. We can be $100(1-\gamma)\%$ confident that the true differences between the population means of the model and system output variables are simultaneously contained within (3).

In approach II, the $100(1-\gamma^m)\%$ SCI are first constructed for $\boldsymbol{\mu}^m$ as

$$[\boldsymbol{\delta}^m, \boldsymbol{\tau}^m] \tag{4}$$

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

$$[\boldsymbol{\delta}^s, \boldsymbol{\tau}^s] \tag{5}$$

where

$$(\boldsymbol{\delta}^s)' = [\delta_1^s, \delta_2^s, \ldots, \delta_k^s] \quad \text{and} \quad (\boldsymbol{\tau}^s)' = [\tau_1^s, \tau_2^s, \ldots, \tau_k^s]$$

Finally, using the Bonferroni inequality, the MRA is determined by the following SCI for $\boldsymbol{\mu}^m - \boldsymbol{\mu}^s$ 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 [95]:

$$[\boldsymbol{\delta}^m - \boldsymbol{\tau}^s, \boldsymbol{\tau}^m - \boldsymbol{\delta}^s] \tag{6}$$

In approach III the model and system output variables are observed in pairs and the MRA is determined by the $100(1-\gamma)\%$ SCI for $\boldsymbol{\mu}^d$, the population means of the differences of paired observations, as

$$[\boldsymbol{\delta}^d, \boldsymbol{\tau}^d] \tag{7}$$

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

The approach for constructing the MRA should be chosen with respect to the way the model is driven. The MRA 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, "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, "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

$$L \le \boldsymbol{\mu}^m - \boldsymbol{\mu}^s \le \mathbf{U} \tag{8}$$

where $\mathbf{L}' = [L_1, L_2, \ldots, L_k]$ and $\mathbf{U}' = [U_1, U_2, \ldots, U_k]$ 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 MRA should be compared against (6) to evaluate model validity.

The shorter the lengths of the MRA, 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 MRA, data collection method, and cost of data collection. For details of performing the trade-off analysis, see ref. 30. The confidence interval validation procedure is presented in Figure 10.11.

Structural Testing. The structural testing (also known as *white-box testing*) category consists of the six techniques discussed below. Structural (white-box) testing is used to evaluate the model based on its internal structure (how it is built), whereas functional (black-box) testing is intended for assessing the input–output transformation accuracy of the model. Structural testing employs data flow and control flow diagrams to assess the accuracy of internal model structure by examining model elements such as statements, branches, conditions, loops, internal logic, internal data representations, submodel interfaces, and model execution paths.

1. *Branch Testing:* conducted to run/exercise the simulation model under test data so as to execute as many branch alternatives as possible, as many times as possible, and to substantiate their accurate operations. The more branches are tested successfully, the more confidence we gain in model's accurate execution with respect to its logical branches [23].

2. *Condition Testing:* conducted to run/exercise the simulation model under test data so as to execute as many (compound) logical conditions as possible, as many times as possible, and to substantiate their accurate operations. The more logical conditions are tested successfully, the more confidence we gain in model's accurate execution with respect to its logical conditions.

3. *Data Flow Testing:* uses the control flow graph to explore sequences of events related to the status of data structures and to examine data-flow anomalies. For example, sufficient paths can be forced to execute under test data to assure that every data element and structure is initialized prior to use or every declared data structure is used at least once in an executed path [23].

4. *Loop Testing:* conducted to run/exercise the simulation model under test data so

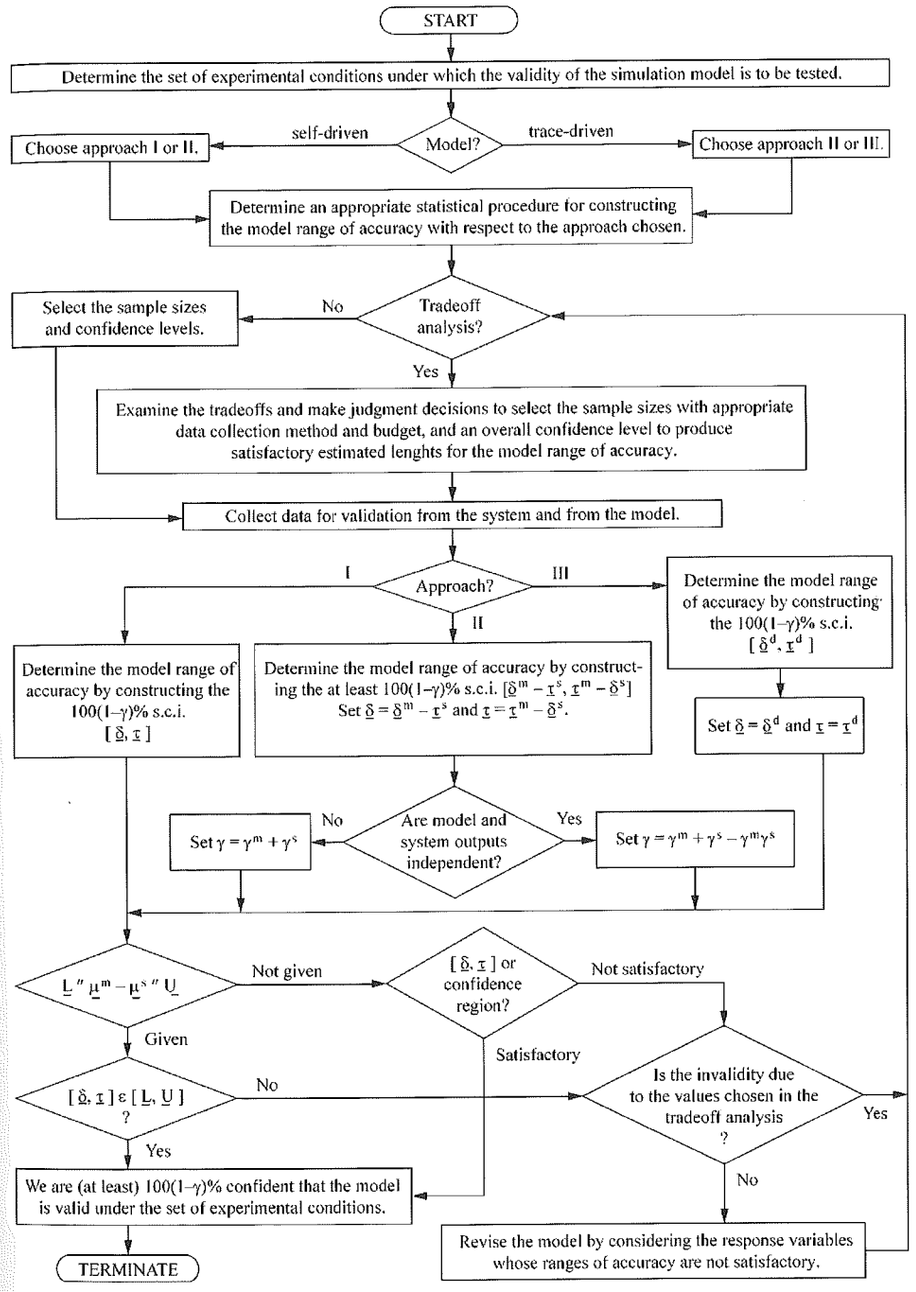

Figure 10.11 Validation procedure using simultaneous confidence intervals.

as to execute as many loop structures as possible, as many times as possible, and to substantiate their accurate operations. The more loop structures are successfully tested, the more confidence we gain in the model's accurate execution with respect to its loop structures [50].

5. *Path Testing:* conducted to run/exercise the simulation model under test data so as to execute as many control flow paths as possible, as many times as possible, and to substantiate their accurate operations. The more control flow paths are tested successfully, the more confidence we gain in model's accurate execution with respect to its control flow paths. However, 100% path coverage is impossible to achieve for a reasonably large simulation model [23].

Path testing is performed in three steps [71]. In step 1 the model control structure is determined and represented in a control flow diagram. In step 2 test data are 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 are 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 [96]. 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 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 [96] prove that the path prefix strategy achieves total branch coverage.

The identification of essential paths is a strategy that reduces the path coverage required by nearly 40% [97]. The basis for the reduction is the elimination of nonessential paths. Paths that are overlapped by other paths are nonessential. The model control flow graph is transformed into a directed graph whose arcs (called *primitive arcs*) correspond to the essential paths of the model. Nonessential 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 [97] presents algorithms for efficiently identifying nonessential 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.

6. *Statement Testing:* conducted to run/exercise the simulation model under test data so as to execute as many statements as possible, as many times as possible, and to substantiate their accurate operations. The more statements are tested successfully, the more confidence we gain in the model's accurate execution with respect to its statements [23].

Submodel/Module Testing. Submodel/module testing requires a top-down model decomposition in terms of submodels/modules. The executable 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.

Validating each submodel individually does not imply sufficient validity for the entire model as dictated by principle 12; each submodel is found sufficiently valid with some allowable error and the allowable errors can accumulate to make the entire model invalid. Therefore, after individually validating each submodel, the entire model itself must be subjected to overall testing.

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," that is, 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. Then the debugger is employed to understand how and why the error occurs.

Current state-of-the-art debuggers (or interactive run-time controllers) allow viewing the run-time 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 that lead up to the errors [42].

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 tested thoroughly. 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 [41].

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 that has no other function than to let its caller complete the call. Fairley [65] 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 are 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) [65].

Visualization/Animation. Visualization/animation of a simulation model greatly assists in model VV&T [24, 98]. Displaying graphical images of internal (e.g., how customers are served by a cashier) and external (e.g., utilization of the cashier) dynamic behavior of a model during execution enables us to discover errors by seeing. For exam-

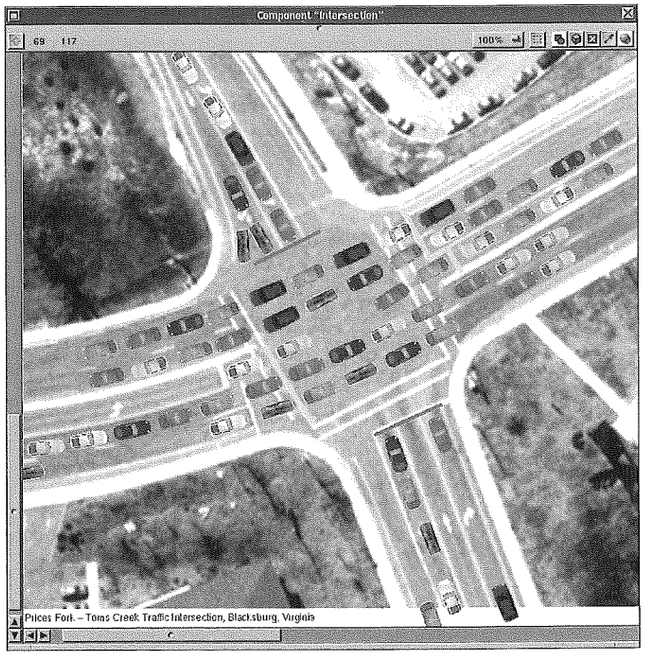

Figure 10.12 Traffic intersection simulation model animation.

ple, in the case study, we can observe the arrivals of vehicles in different lanes and their movements through the intersection as the traffic light changes, as shown in Figure 10.12. Seeing the animation 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. In the case study, the animation was extremely useful for identifying bugs in the model logic. Many errors were reflected in the animation and were easily noticed.

Seeing the model in action is very useful for uncovering errors; however, seeing is not believing in visual simulation [99]. Observing that the animation of model behavior is free of errors does not guarantee the correctness of the model results.

10.4.4 Formal VV&T Techniques

Formal VV&T techniques are based on mathematical proof of correctness. If attainable, 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 proof of correctness techniques are simply not capable of being applied even to a reasonably complex simulation model. However, formal techniques serve as the foundation for other VV&T techniques and the most commonly known eight techniques are described briefly below: (1) induction, (2) inductive assertions, (3) inference, (4) λ-calculus, (5) logical deduction, (6) predicate calculus, (7) predicate transformation, and (8) proof of correctness [42, 100].

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). 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, and so on, the model must be correct if it terminates. Formal induction proof techniques exist for the intuitive explanation just given.

Birta and Özmizrak [101] present a knowledge-based approach for simulation model validation based on the use of a validation knowledge base containing rules of inference.

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, that is, 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, 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 [102, 103].

The *λ-calculus* [104] 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 [105].

Predicate transformation [106, 107] provides a basis for verifying model correctness by formally defining the semantics of the model with a mapping that 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 [37, 105]. 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.

10.5 CREDIBILITY ASSESSMENT STAGES

It is very important to understand the 15 principles of VV&T presented in Section 10.3 when applying more than 75 VV&T techniques described in Section 10.4 throughout

the entire life cycle of a simulation study given in Figure 10.1. The principles help the researchers, practitioners, and managers better understand what VV&T is all about. These principles serve to provide the underpinnings for the VV&T techniques. Understanding and applying the principles is crucially important for the success of a simulation study.

Table 10.4 marks the VV&T techniques that are applicable for each major credibility assessment stage of the life cycle of a simulation study. The rows of Table 10.4 list the VV&T techniques in alphabetical order. The column labels correspond to the major credibility assessment stages in the life cycle:

- Formulated problem VV&T
- Feasibility assessment of simulation
- System and objectives definition VV&T
- Model qualification
- Communicative model VV&T
- Programmed model VV&T
- Experiment design VV&T
- Data VV&T
- Experimental model VV&T
- Presentation VV&T

It should be noted that the list above shows only the major credibility assessment stages and that many other VV&T activities exist throughout the life cycle.

10.5.1 Formulated Problem VV&T

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 [9]. Failure to formulate the *actual* problem results in a type III error. Once a type III error is committed, regardless of how well the problem is solved, the simulation study will either end unsuccessfully or with a type II error. Therefore, the accuracy of the formulated problem greatly affects the credibility and acceptability of simulation results.

In the case study, type III error may be committed if the problem domain boundary excludes the adjacent traffic intersections. It is possible that the traffic light timings of the adjacent intersections are set in such a way that they all turn green at the same time for the traffic traveling toward the intersection under study. Such light timings may be the root cause of congestion. Correcting the light timings at the adjacent traffic intersections may very well solve the congestion problem at the traffic intersection under study. Failure to identify such a cause may result in type III error.

Audit, cause–effect graphing, desk checking, face validation, inspections, reviews, 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 [9]. The questionnaire developed by Balci and Nance [9] with 38 indicators can be used in applying audit, inspections, reviews, and walkthroughs.

TABLE 10.4 Applicability of the VV&T Techniques for the Credibility Assessment Stages

	FP VV&T	FA of Simulation	S&OD VV&T	Model Qualification	CM VV&T	PM VV&T	ED VV&T	Data VV&T	EM VV&T	Presentation VV&T
Acceptance testing									×	
Alpha testing									×	
Assertion checking						×	×	×	×	
Audit		×	×	×	×	×	×	×	×	×
Authorization testing						×			×	
Beta testing									×	
Bottom-up testing						×			×	
Boundary value testing						×			×	
Branch testing						×	×		×	
Calling structure analysis					×	×			×	
Cause–effect graphing		×			×	×			×	
Comparison testing									×	
Concurrent process analysis					×	×			×	
Condition testing						×	×		×	
Control flow analysis					×	×	×		×	
Data dependency analysis					×	×	×	×	×	
Data flow analysis					×	×	×	×	×	
Data flow testing						×	×		×	
Data interface testing								×	×	
Debugging						×	×		×	
Desk checking		×	×	×	×	×	×	×	×	×
Documentation checking		×	×	×	×	×	×	×	×	
Equivalence partitioning testing						×			×	
Execution monitoring						×	×		×	
Execution profiling						×	×		×	

TABLE 10.4 (*Continued*)

	FP VV&T	FA of Simulation	S&OD VV&T	Model Qualification	CM VV&T	PM VV&T	ED VV&T	Data VV&T	EM VV&T	Presentation VV&T
Execution tracing						×	×		×	
Extreme input testing						×			×	
Face validation	×	×	×	×	×	×	×	×	×	×
Fault/failure analysis					×	×	×	×	×	
Fault/failure insertion testing						×	×	×	×	
Field testing									×	
Functional testing						×	×		×	
Graphical comparisons						×	×		×	
Induction						×			×	
Inductive assertions						×			×	
Inference						×			×	
Inspections	×	×	×	×	×	×	×	×	×	×
Invalid input testing						×			×	
Lambda calculus						×			×	
Logical deduction						×			×	
Loop testing						×	×		×	
Model interface analysis					×	×			×	
Model interface testing						×			×	
Object-flow testing						×			×	
Partition testing						×	×		×	
Path testing						×	×		×	
Performance testing									×	
Predicate calculus						×			×	
Predicate transformation						×			×	
Predictive validation									×	
Product testing									×	
Proof of correctness						×			×	
Real-time input testing						×			×	
Regression testing						×			×	
Reviews	×	×	×	×	×	×	×	×	×	×
Security testing									×	
Self-driven input testing						×			×	
Semantic analysis						×	×		×	
Sensitivity analysis						×	×		×	
Standards testing						×			×	
State transition analysis					×	×	×		×	
Statement testing						×	×		×	
Statistical techniques (Table 10.3)								×	×	
Stress testing						×			×	
Structural analysis					×	×	×		×	
Submodel/module testing						×	×		×	
Symbolic debugging						×	×		×	
Symbolic evluation					×	×	×		×	
Syntax analysis						×	×		×	
Top-down testing						×	×		×	
Trace-driven input testing						×			×	
Traceability assessment					×	×			×	
Turing test									×	
User interface analysis					×	×			×	
User interface testing						×			×	
Visualization/animation						×	×	×	×	×
Walkthroughs	×	×	×	×	×	×	×	×	×	×

10.5.2 Feasibility Assessment of Simulation

Audit, desk checking, 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?

10.5.3 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.

Audit, desk checking, face validation, inspections, reviews, 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 six 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.

10.5.4 Model Qualification

Model qualification is intended for assessing the credibility of the model formulation process. A model should be conceptualized under the guidance of a structured approach such as the conical methodology [5]. One key idea behind the use of a structured approach is to control the model complexity so that we can verify and validate the model successfully. 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 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, desk checking, face validation, inspections, reviews, and walkthroughs can be applied for conducting model qualification.

In the case study, many assumptions including the following, are made in abstracting the traffic intersection operation: (1) pedestrians are excluded; (2) bicycles and emergency vehicles are excluded; (3) the light timing cycle length is assumed constant and the sensor in lane 9 is ignored; (4) the yellow light is included in the green since most drivers pass on yellow; (5) all drivers obey the traffic laws; and (6) all vehicles have the same size. These assumptions were justified to be appropriate under the study objectives in the model qualification credibility assessment stage.

10.5.5 Communicative Model VV&T

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* [25] is the prescribed conditions for which the model is intended to match the system under study. *Level of agreement* [25] is the required correspondence between the model and the system, consistent with the domain of intended application and the study objectives.

In the case study, the graphical model design specification, shown in Figure 10.3, is justified to be sufficiently accurate. Inspections are conducted to substantiate that all vehicle movements in the model design specification represent the real-life movements with sufficient accuracy. Specifications of all classes is found to be appropriate.

10.5.6 Programmed Model VV&T

Programmed model VV&T deals with the assessment of programmed (executable) model accuracy. Most of the techniques in Table 10.4 are applicable for conducting programmed model VV&T. In the case study, many of the applicable techniques in Table 10.4 were used to assess the executable model accuracy. Specifically, the animation was very helpful. In addition, tracing of message passing was instrumental in revealing some of the bugs.

10.5.7 Experiment Design VV&T

Experiment design VV&T deals with substantiating the sufficient accuracy of the design of experiments. The techniques marked in Table 10.4 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., $X = -\log_e(1-U)$) [108]); (3) How well is the random number generator tested? (using a generator that is not rigorously shown to produce uniformly distributed independent numbers with sufficiently large period may invalidate the entire experiment design); (4) Are *appropriate* statistical techniques implemented to design and analyze the simulation experiments? How well are the underlying assumptions satisfied? (see ref. 109 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 startup problem) [110] appropriately addressed? and (6) For comparison studies, are identical experimental conditions replicated correctly for each of the alternative operating policies compared?

10.5.8 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 10.1 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 [111] 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.

The techniques marked in Table 10.4 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 [108]); and (6) Are all data up to date?

10.5.9 Experimental Model VV&T

Experimental model VV&T deals with substantiating that the experimental model has sufficient accuracy in representing the system under study consistent with the study objectives. All of the techniques listed in Table 10.4 can be applied for conducting experimental model VV&T. The applicability of the VV&T techniques depends on 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 10.3 are applicable only for case 1.

In the case study, many of the applicable techniques in Table 10.4 were used to assess the experimental model accuracy. Some of the statistical techniques in Table 10.3 were also used.

10.5.10 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. 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 on 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 [112] points out that "we do not know of any model assessment or modeling project review that indicated satisfaction with the available documentation." Nance [5] 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 [16, 29, 113, 114] 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 10.1.

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, desk checking, face validation, inspections, reviews, visualization/animation, and walkthroughs can be applied for conducting presentation VV&T.

10.6 CONCLUDING REMARKS

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 of a simulation study.

Assessing credibility throughout the life cycle 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 credibility assessment is required to alleviate these problems. More research is needed to bring automation to the application of VV&T techniques.

Integration VV&T with model development is crucial. This integration is best achieved within a computer-aided simulation model development environment [16, 29, 113, 114]. More research is needed for this integration. The question of which of the applicable VV&T techniques should be selected for a particular VV&T activity in the life cycle should be answered by taking the following into consideration: (1) model type, (2) simulation type, (3) problem domain, and (4) study objectives.

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 [37].

Subjectivity is, and always will be, part of the credibility assessment for a reasonably

complex simulation study. The reason for subjectivity is twofold: 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.

REFERENCES

1. Ören, T. I. (1981). Concepts and criteria to assess acceptability of simulation studies: a frame of reference, *Communications of the ACM*, Vol. 24, No. 4, pp. 180–189.
2. 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.
3. Ören, T. I. (1987). Quality assurance paradigms for artificial intelligence in modelling and simulation," *Simulation*, Vol. 48, No. 4, pp. 149–151.
4. 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, N.J., pp. 25–32.
5. Nance, R. E. (1994). The conical methodology and the evolution of simulation model development, *Annals of Operations Research*, Vol. 53, pp. 1–46.
6. 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., Society for Computer Simulation, San Diego, Calif., pp. 13–18.
7. Knepell, P. L., and D. C. Arangno (1993). Simulation validation: a confidence assessment methodology, *Monograph 3512-04*, IEEE Computer Society Press, Los Alamitos, Calif.
8. Woolley, R. N., and M. Pidd (1981). Problem structuring: a literature review, *Journal of the Operational Research Society*, Vol. 32, No. 3, pp. 197–206.
9. Balci, O., and R. E. Nance (1985). Formulated problem verification as an explicit requirement of model credibility," *Simulation*, Vol. 45, No. 2, pp. 76–86.
10. Shannon, R. E. (1975). *Systems Simulation: The Art and Science*, Prentice Hall, Upper Saddle River, N.J.
11. Law, A. M., and W. D. Kelton (1991). *Simulation Modeling and Analysis*, 2nd ed., McGraw-Hill, New York.
12. Banks, J., J. S. Carson, and B. L. Nelson (1996). *Discrete-Event System Simulation*, 2nd ed., Prentice Hall, Upper Saddle River, N.J.
13. Vincent, S., and A. M. Law (1995). ExpertFit: total support for simulation input modeling," in *Proceedings of the 1995 Winter Simulation Conference*, C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, eds., IEEE, Piscataway, N.J., pp. 395–400.
14. Overstreet, C. M., and R. E. Nance (1985). A specification language to assist in analysis of discrete event simulation models, *Communications of the ACM*, Vol. 28, No. 2, pp. 190–201.
15. Martin, J., and C. McClure (1985). *Diagramming Techniques for Analysts and Programmers*, Prentice Hall, Upper Saddle River, N.J.
16. Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance (1995). A picture-based object-oriented visual simulation environment, in *Proceedings of the 1995 Winter Simulation Conference*, C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, eds., IEEE, Piscataway, N.J., pp. 1333–1340.
17. Orca Computer (1996). *Visual Simulation Environment User's Guide*, Orca Computer, Inc., Blacksburg, Va.

18. Orca Computer (1996). *Visual Simulation Environment Reference Manual*, Orca Computer, Inc., Blacksburg, Va.
19. Banks, J. (1996). Software for Simulation, in *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, eds., IEEE, Piscataway, N.J., pp. 31–38.
20. 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, N.J., pp. 287–295.
21. Fishman, G. S. (1978). *Principles of Discrete Event Simulation*, Wiley-Interscience, New York.
22. Hetzel, W. (1984). *The Complete Guide to Software Testing*, QED Information Sciences, Wellesley, Mass.
23. Beizer, B. (1990). *Software Testing Techniques*, 2nd ed., Van Nostrand Reinhold, New York.
24. Sargent, R. G. (1996). Verifying and validating simulation models, in *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, eds., IEEE, Piscataway, N.J., pp. 55–64.
25. Schlesinger, S., et al. (1979). Terminology for model credibility, *Simulation*, Vol. 32, No. 3, pp. 103–104.
26. Balci, O., and R. G. Sargent (1981). A methodology for cost-risk analysis in the statistical validation of simulation models, *Communications of the ACM*, Vol. 24, No. 4, pp. 190–197.
27. Nance, R. E., and C. M. Overstreet (1987). Diagnostic assistance using digraph representations of discrete event simulation model specifications, *Transactions of the SCS*, Vol. 4, No. 1, pp. 33–57.
28. Balci, O., and R. E. Nance (1992). The simulation model development environment: an overview, in *Proceedings of the 1992 Winter Simulation Conference*, J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, eds., IEEE, Piscataway, N.J., pp. 726–736.
29. Derrick, E. J., and O. Balci (1995). A visual simulation support environment based on the DOMINO conceptual framework, *Journal of Systems and Software*, Vol. 31, No. 3, pp. 215–237.
30. Balci, O., and R. G. Sargent (1984). Validation of simulation models via simultaneous confidence intervals, *American Journal of Mathematical and Management Sciences*, Vol. 4, No. 3–4, pp. 375–406.
31. Johnson, M. E., and M. Mollaghasemi (1994). Simulation input data modeling, *Annals of Operations Research*, Vol. 53, pp. 47–75.
32. Watson, C. E. (1976). The problems of problem solving, *Business Horizons*, Vol. 19, No. 4, pp. 88–94.
33. Perry, W. (1995). *Effective Methods for Software Testing*, Wiley, New York.
34. 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, New York, pp. 211–266.
35. Adrion, W. R., M. A. Branstad, and J. C. Cherniavsky (1982). Validation, verification, and testing of computer software, *Computing Surveys*, Vol. 14, No. 2, pp. 159–192.
36. Hermann, C. F. (1967). Validation problems in games and simulations with special reference to models of international politics, *Behavioral Science*, Vol. 12, No. 3, pp. 216–231.
37. Schach, S. R. (1996). *Software Engineering*, 3rd ed., Richard D. Irwin, Homewood, Ill.
38. 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, Germany, September 25–30, H.-L. Hausen, ed., pp. 13–40.

39. 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, New York, pp. 137–177.
40. Knight, J. C., and E. A. Myers (1993). An improved inspection technique, *Communications of the ACM*, Vol. 36, No. 11, pp. 51–61.
41. Sommerville, I. (1996). *Software Engineering*, 5th ed., Addison-Wesley, Reading, Mass.
42. 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, N.J., pp. 559–568.
43. Schruben, L. W. (1980). Establishing the credibility of simulations, *Simulation*, Vol. 34, No. 3, pp. 101–105.
44. Turing, A. M. (1963). Computing machinery and intelligence, in *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, pp. 11–15.
45. Van Horn, R. L. (1971). Validation of simulation results, *Management Science*, Vol. 17, No. 5, pp. 247–258.
46. Deutsch, M. S. (1982). *Software Verification and Validation: Realistic Project Approaches*, Prentice Hall, Upper Saddle River, N.J.
47. Myers, G. J. (1978). A controlled experiment in program testing and code walkthroughs/inspections, *Communications of the ACM*, Vol. 21, No. 9, pp. 760–768.
48. Myers, G. J. (1979). *The Art of Software Testing*, Wiley, New York.
49. Yourdon, E. (1985). *Structured Walkthroughs*, 3rd ed., Yourdon Press, New York.
50. Pressman, R. S. (1996). *Software Engineering: A Practitioner's Approach*, 4th ed., McGraw-Hill, New York.
51. Miller, L. A., E. H. Groundwater, J. E. Hayes, and S. M. Mirsky (1995). Survey and assessment of conventional software verification and validation methods, *Special Publication NUREG/CR-6316*, Vol. 2, U.S. Nuclear Regulatory Commission, Washington, DC.
52. Rattray, C., ed. (1990). *Specification and Verification of Concurrent Systems*, Springer-Verlag, New York.
53. 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, pp. 407–421.
54. Dunn, R. H. (1984). *Software Defect Removal*, McGraw-Hill, New York.
55. Allen, F. E., and J. Cocke (1976). A program data flow analysis procedure, *Communications of the ACM*, Vol. 19, No. 3, pp. 137–147.
56. 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, N.J., pp. 783–790.
57. Yücesan, E., and S. H. Jacobson (to appear). Computational issues for accessibility in discrete event simulation, *ACM Transactions on Modeling and Computer Simulation*, Vol. 6, No. 1, pp. 53–75.
58. King, J. C. (1976). Symbolic execution and program testing, *Communications of the ACM*, Vol. 19, No. 7, pp. 385–394.
59. Dillon, L. K. (1990). Using symbolic execution for verification of Ada Tasking programs, *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 4, pp. 643–669.
60. Ramamoorthy, C. V., S. F. Ho, and W. T. Chen (1976). On the automated generation of program test data, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 293–300.
61. 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, Upper Saddle River, N.J., pp. 80–111.

62. Department of Defense (1995). Modeling and simulation (M&S) master plan, *DoD 5000.59-P*, October.
63. 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, New York, pp. 342–384.
64. Fairley, R. E. (1975). An experimental program-testing facility, *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, pp. 350–357.
65. 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, Calif., pp. 708–710.
66. Howden, W. E. (1980). Functional program testing, *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 2, pp. 162–169.
67. Cohen, K. J., and R. M. Cyert (1961). Computer models in dynamic economics, *Quarterly Journal of Economics*, Vol. 75, No. 1, 112–127.
68. Forrester, J. W. (1961). *Industrial Dynamics*, MIT Press, Cambridge, Mass.
69. Miller, D. K. (1975). Validation of computer simulations in the social sciences, in *Proceedings of the 6th Annual Conference on Modeling and Simulation*, Pittsburgh, Pa., pp. 743–746.
70. 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, Calif., July 14–16, Simulation Councils, La Jolla, Calif., pp. 1286–1296.
71. Howden, W. E. (1976). Reliability of the path analysis testing strategy, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 3, pp. 208–214.
72. Richardson, D. J., and L. A. Clarke (1985). Partition analysis: a method combining testing and verification, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12, pp. 1477–1490.
73. Emshoff, J. R., and R. L. Sisson (1970). *Design and Use of Computer Simulation Models*, Macmillan, New York.
74. Miller, D. R. (1974). Model validation through sensitivity analysis, in *Proceedings of the 1974 Summer Computer Simulation Conference*, Houston, Texas, July 9–11, Simulation Councils, La Jolla, Calif., pp. 911–914.
75. Miller, D. R. (1974). Sensitivity analysis and validation of simulation models, *Journal of Theoretical Biology*, Vol. 48, No. 2, pp. 345–360.
76. Ould, M. A., and C. Unwin (1986). *Testing in Software Development*, Cambridge University Press, Cambridge.
77. Naylor, T. H., and J. M. Finger (1967). Verification of computer simulation models, *Management Science*, Vol. 14, No. 2, pp. B92–B101.
78. Balci, O., and R. G. Sargent (1982). 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, N.J., pp. 620–629.
79. Balci, O., and R. G. Sargent (1982). Validation of multivariate response models using Hotelling's two-sample T^2 test, *Simulation*, Vol. 39, No. 6, pp. 185–192.
80. 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. pp. 309–323.
81. Garratt, M. (1974). Statistical validation of simulation models, in *Proceedings of the 1974 Summer Computer Simulation Conference*, Houston, Texas, July 9–11, Simulation Councils, La Jolla, Calif., pp. 915–926.

82. 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*, Wiley, New York, pp. 702–705.

83. Aigner, D. J. (1972). A note on verification on computer simulation models, *Management Science*, Vol. 18, No. 11, pp. 615–619.

84. 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, N.C., pp. 207–231.

85. Kheir, N. A., and W. M. Holmes (1978). On validating simulation models of missile systems, *Simulation*, Vol. 30, No. 4, pp. 117–128.

86. Rowland, J. R., and W. M. Holmes (1978). Simulation validation with sparse random data, *Computers and Electrical Engineering*, Vol. 5, No. 3, pp. 37–49.

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

88. Fishman, G. S., and P. J. Kiviat (1967). The analysis of simulation generated time series, *Management Science*, Vol. 13, No. 7, pp. 525–557.

89. Gallant, A. R., T. M. Gerig, and J. W. Evans (1974). Time series realizations obtained according to an experimental design, *Journal of the American Statistical Association*, Vol. 69, No. 347, pp. 639–645.

90. Hunt, A. W. (1970). Statistical evaluation and verification of digital simulation models through spectral analysis, Ph.D. dissertation, University of Texas at Austin, Austin, Texas.

91. Watts, D. (1969). Time series analysis, in *The Design of Computer Simulation Experiments*, T. H. Naylor, ed., Duke University Press, Durham, N.C., pp. 165–179.

92. Damborg, M. J., and L. F. Fuller (1976). Model validation using time and frequency domain error measures, *ERDA Report 76-152*, NTIS, Springfield, Va.

93. 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 Arsenol, Ala., June.

94. Teorey, T. J. (1975). Validation criteria for computer system simulations, *Simuletter*, Vol. 6, No. 4, pp. 9–20.

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

96. Prather, R. E., and J. P. Myers, Jr. (1987). The path prefix software testing strategy, *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 7, pp. 761–766.

97. Chusho, T. (1987). Test data selection and quality estimation based on the concept of essential branches for path testing, *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, pp. 509–517.

98. Bell, P. C., and R. M. O'Keefe (1994). Visual interactive simulation: a methodological perspective, *Annals of Operations Research*, Vol. 53, pp. 321–342.

99. 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, pp. 422–432.

100. Khanna, S. (1991). Logic programming for software verification and testing, *The Computer Journal*, Vol. 34, No. 4, pp. 350–357.

101. Birta, L. G., and F. N. Özmizrak (1996). A knowledge-based approach for the validation of simulation models: the foundation, *ACM Transactions on Modeling and Computer Simulation*, Vol. 6, No. 1, pp. 76–98.

102. Manna, Z., S. Ness, and J. Vuillemin (1973). Inductive methods for proving properties of programs, *Communications of the ACM*, Vol. 16, No. 8, pp. 491–502.

103. Reynolds, C., and R. T. Yeh (1976). Induction as the basis for program verification, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 244–252.

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

105. Backhouse, R. C. (1986). *Program Construction and Verification*, Prentice Hall International, London.

106. Dijkstra, E. W. (1975). Guarded commands, non-determinacy and a calculus for the derivation of programs, *Communications of the ACM*, Vol. 18, No. 8, pp. 453–457.

107. Yeh, R. T. (1977). Verification of programs by predicate transformation, in *Current Trends in Programming Methodology*, Vol. 2, R. Yeh, ed., Prentice Hall, Upper Saddle River, N.J., pp. 228–247.

108. 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, N.J., pp. 227–242.

109. Law, A. M. (1983). Statistical analysis of simulation output data, *Operations Research*, Vol. 31, No. 6, pp. 983–1029.

110. Wilson, J. R., and A. A. B. Pritsker (1978). A survey of research on the simulation startup problem, *Simulation*, Vol. 31, No. 2, pp. 55–58.

111. U.S. GAO (1987). DOD simulations: improved assessment procedures would increase the credibility of results, *GAO/PEMD-88-3*, U.S. General Accounting Office, Washington, D.C., Dec.

112. Gass, S. I. (1983). Decision-aiding models: validation, assessment, and related issues for policy analysis, *Operations Research*, Vol. 31, No. 4, pp. 603–631.

113. Balci, O. (1986). Requirements for model development environments, *Computers and Operations Research*, Vol. 13, No. 1, pp. 53–67.

114. Balci, O., and R. E. Nance (1987). Simulation model development environments: a research prototype, *Journal of the Operational Research Society*, Vol. 38, No. 8, pp. 753–763.