1

Basic Concepts and Techniques

This chapter introduces the foundations of modeling and simulation, and elucidates their importance. We survey the different approaches to modeling that are used in practice and discuss at a high level the methodology that should be followed when executing a modeling project.

1.1 Why is Simulation Important?

Simulation is the imitation of a real-world system through a computational re-enactment of its behavior according to the rules described in a mathematical model.[1] Simulation serves to imitate a real system or process. The act of simulating a system generally entails considering a limited number of key characteristics and behaviors within the physical or abstract system of interest, which is otherwise infinitely complex and detailed. A simulation allows us to examine the system's behavior under different scenarios, which are assessed by re-enactment within a virtual computational world. Simulation can be used, among other things, to identify bottlenecks in a process, provide a safe and relatively cheaper (in term of both cost and time) test bed to evaluate the side effects, and optimize the performance of the system—all before realizing these systems in the physical world.

Early in the twentieth century, modeling and simulation played only a minor role in the system design process. Having few alternatives, engineers moved straight from paper designs to production so they could test their designs. For example, when Howard Hughes wanted to build a new aircraft, he never knew if it would fly until it was

[1] Alan Turing used the term *simulation* to refer to what happens when a digital computer runs a finite state machine (FSM) that describes the state transitions, inputs, and outputs of a subject discrete-state system. In this case, the computer simulates the subject system by executing a corresponding FSM.

Network Modeling and Simulation M. Guizani, A. Rayes, B. Khan and A. Al-Fuqaha

actually built. Redesigns meant starting from scratch and were very costly. After World War II, as design and production processes became much more expensive, industrial companies like Hughes Aircraft collapsed due to massive losses. Clearly, new alternatives were needed. Two events initiated the trend toward modeling and simulation: the advent of computers; and NASA's space program. Computing machinery enabled large-scale calculations to be performed quickly. This was essential for the space program, where projections of launch and re-entry were critical. The simulations performed by NASA saved lives and millions of dollars that would have been lost through conventional rocket testing. Today, modeling and simulation are used extensively. They are used not just to find if a given system design works, but to discover a system design that works best. More importantly, modeling and simulation are often used as an inexpensive technique to perform exception and "what-if" analyses, especially when the cost would be prohibitive when using the actual system. They are also used as a reasonable means to carry out stress testing under exceedingly elevated volumes of input data.

Japanese companies use modeling and simulation to improve quality, and often spend more than 50% of their design time in this phase. The rest of the world is only now beginning to emulate this procedure. Many American companies now participate in rapid prototyping, where computer models and simulations are used to quickly design and test product concept ideas before committing resources to real-world in-depth designs.

Today, simulation is used in many contexts, including the modeling of natural systems in order to gain insight into their functioning. Key issues in simulation include acquisition of valid source information about the referent, selection of key characteristics and behaviors, the use of simplifying approximations and assumptions within the simulation, and fidelity and validity of the simulation outcomes. Simulation enables goal-directed experimentation with dynamical systems, i.e., systems with time-dependent behavior. It has become an important technology, and is widely used in many scientific research areas.

In addition, modeling and simulation are essential stages in the engineering design and problem-solving process and are undertaken before a physical prototype is built. Engineers use computers to draw physical structures and to make mathematical models that simulate the operation of a device or technique. The modeling and simulation phases are often the longest part of the engineering design process. When starting this phase, engineers keep several goals in mind:

- Does the product/problem meet its specifications?
- What are the limits of the product/problem?
- Do alternative designs/solutions work better?

The modeling and simulation phases usually go through several iterations as engineers test various designs to create the best product or the best solution to a

Table 1.1 Examples of simulation usage in training and education

Usage of simulation	Examples
Training to enhance motor and operational skills (and associated decision-making skills)	• Virtual simulation which uses virtual equipment and real people (human-in-the-loop) in a simulation study • Aircraft simulator for pilot training • Augmented reality simulation (such as in-flight pilot training with additional artificial intelligence aircraft) • Virtual body for medicine • Nuclear reactor simulator • Power plant simulator • Simulators for the selection of operators (such as pilots) • Live simulation (use of simulated weapons along with real equipment and people)
Training to enhance decision-making skills	• Constructive simulation (war game simulation) • Simulation for operations other than war (non-article 5 operations, in NATO terminology): peace support operations; conflict management (between individuals, groups, nations) • Business game simulations • Agent-based simulations • Holonic agent simulations that aim to explore benefits of cooperation between individuals, companies (mergers), and nations
Education	• Simulation for the teaching/learning of dynamic systems (which may have trajectory and/or structural behavior): simulation of adaptive systems, time-varying systems, evolutionary systems, etc.

problem. Table 1.1 provides a summary of some typical uses of simulation modeling in academia and industry to provide students and professionals with the right skill set.

Simulation provides a method for checking one's understanding of the world and helps in producing better results faster. A simulation environment like MATLAB is an important tool that one can use to:

- Predict the course and results of certain actions.
- Understand why observed events occur.
- Identify problem areas before implementation.
- Explore the effects of modifications.

- Confirm that all variables are known.
- Evaluate ideas and identify inefficiencies.
- Gain insight and stimulate creative thinking.
- Communicate the integrity and feasibility of one's plans.

One can use simulation when the analytical solution does not exist, is too complicated, or requires more computational time than the simulation. Simulation should not be used in the following cases:

- The simulation requires months or years of CPU time. In this scenario, it is probably not feasible to run simulations.
- The analytical solution exists and is simple. In this scenario, it is easier to use the analytical solution to solve the problem rather than use simulation (unless one wants to relax some assumptions or compare the analytical solution to the simulation).

1.2 What is a Model?

A computer model is a computer program that attempts to simulate an abstract model of a particular system. Computer models can be classified according to several orthogonal binary criteria including:

- **Continuous state or discrete state:** If the state variables of the system can assume any value, then the system is modeled using a *continuous-state model*. On the other hand, a model in which the state variables can assume only discrete values is called a *discrete-state model*. Discrete-state models can be continuous- or discrete-time models.
- **Continuous-time or discrete-time models:** If the state variables of the system can change their values at any instant in time, then the system can be modeled by a *continuous-time model*. On the other hand, a model in which the state variables can change their values only at discrete instants of time is called a *discrete-time model*.

A continuous simulation uses differential equations (either partial or ordinary), implemented numerically. Periodically, the simulation program solves all the equations, and uses the numbers to change the state and output of the simulation. Most flight and racing-car simulations are of this type. It may also be used to simulate electric circuits. Originally, these kinds of simulations were actually implemented on analog computers, where the differential equations could be represented directly by various electrical components such as op-amps. By the late 1980s, however, most "analog" simulations were run on conventional digital computers that emulated the behavior of an analog computer.

A discrete-time event-driven simulation (DES) manages events in time [3]. In this type of simulation, the simulator maintains a queue of events sorted by the simulated time they should occur. The simulator reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time. It is often more important to be able to access the data produced by the simulation, to discover logic defects in the design or the sequence of events. Most computer, logic test and fault-tree simulations are of this type.

A special type of discrete simulation that does not rely on a model with an underlying equation, but can nonetheless be represented formally, is *agent-based simulation*. In agent-based simulation, the individual entities (such as molecules, cells, trees, or consumers) in the model are represented directly (rather than by their density or concentration) and possess an internal state and set of behaviors or rules which determine how the agent's state is updated from one time step to the next.

- **Deterministic or probabilistic:** In a *deterministic model*, repetition of the same input will always yield the same output. On the other hand, in a *probabilistic model*, repetition of the same input may lead to different outputs. Probabilistic models use random number generators to model the chance or random events; they are also called *Monte Carlo (MC) simulations* (this is discussed in detail in Chapter 4). *Chaotic models* constitute a special case of deterministic continuous-state models, in which the output is extremely sensitive to input values.
- **Linear or nonlinear models:** If the outputs of the model are linearly related to the inputs, then the model is called a *linear model*. However, if the outputs are not linear functions of the inputs, then the model is called a *nonlinear model*.
- **Open or closed models:** If the model has one or more external inputs, then the model is called an *open model*. On the other hand, the model is called a *closed model* if it has no external inputs at all.
- **Stable or unstable models:** If the dynamic behavior of the model comes to a steady state with time, then the model is called *stable*. Models that do not come to a steady state with time are called *unstable models*. Note that stability refers to time, while the notion of chaos is related to behavioral sensitivities to the input parameters.
- **Local or distributed:** If the model executes only on a single computer then it is called a *local model*. *Distributed models*, on the other hand, run on a network of interconnected computers, possibly in a wide area, over the Internet. Simulations dispersed across multiple host computers are often referred to as *distributed simulations*. There are several military standards for distributed simulation, including Aggregate Level Simulation Protocol (ALSP), Distributed Interactive Simulation (DIS), and the High Level Architecture (HLA).

1.2.1 Modeling and System Terminology

The following are some of the important terms that are used in simulation modeling:

- **State variables:** The variables whose values describe the state of the system. They characterize an attribute in the system such as level of stock in an inventory or number of jobs waiting for processing. In the case where the simulation is interrupted, it can be completed by assigning to the state variables the values they held before interruption of the simulation.
- **Event:** An occurrence at a point in time, which may change the state of the system, e.g., the arrival of a customer or start of work on a job.
- **Entity:** An object that passes through the system, such as cars at an intersection or orders in a factory. Often events (e.g., arrival) are associated with interactions between one or more entities (e.g., customer and store), with each entity having its own state variables (e.g., customer's cash and store's inventory).
- **Queue:** A queue is a linearly ordered list, e.g., a physical queue of people, a task list, a buffer of finished goods waiting for transportation. In short, a place where entities are waiting for something to happen for some coherent reason.
- **Creating:** Causing the arrival of a new entity to the system at some point in time. Its dual process will be referred to as *killing* (the departure of a state).
- **Scheduling:** The act of assigning a new future event to an existing entity.
- **Random variable:** A random variable is a quantity that is uncertain, such as the arrival time between two incoming flights or the number of defective parts in a shipment.
- **Random variate:** A random variate is an artificially generated random variable.
- **Distribution:** A distribution is the mathematical law that governs the probabilistic features of a random variable. Examples of frequently used distributions are discussed in Chapter 7.

1.2.2 Example of a Model: Electric Car Battery Charging Station

Let us discuss the process of building a simulation of a charging station for electric cars at a single station served by a single operative. We assume that the arrival of electric cars as well as their service times are random. First, we have to identify:

- **Entities:** The entities are of three types: cars (multiple instances), the charging station server (single instance), and the traffic generator (single instance) which creates cars and sends them to the battery charging station.
- **States:** The charging station has a server that is in either idle or free mode. Each car C_i has a battery of some capacity K_i and some amount of power P_i present within it. The traffic generator has parameters that specify the distribution of random inter-creation times of cars.

- **Events:** One kind of event is "service car," which is an instruction to the server to charge the next car's battery. Another kind of event is the arrival event, which is sent by the traffic generator to the charging station to inform it that a car has arrived. Finally, there is the "create car event," which is sent by the traffic generator to itself to trigger the creation of a car.
- **Queue:** The queue of cars Q in front of the charging station.
- **Random realizations:** Car inter-creation times, car battery capacities, battery charge levels (and, consequently, car service times), etc.
- **Distributions:** Assume uniform distributions for the car inter-creation times, battery capacities, and charge levels.

Next, we specify what to do at each event.

The traffic generator sends itself a car creation event. Upon receipt of this event, it creates a car with a battery capacity that is randomly chosen uniformly in the interval $[0...F]$, and the battery is set to be $L\%$ full, where L is randomly chosen in the interval $[0...100]$. The traffic generator encapsulates the car in a car arrival event and sends the event to the charging station. It also sends itself another car creation event for some time (chosen uniformly from an interval $[0...D]$ minutes) in the future.

When a car arrival event is received at the charging station, the car is added to the end of the queue. If (at the moment just prior to this occurrence) the server was in free mode, and the queue transitioned from empty to non-empty, then the server is sent a service car event.

The server, upon receiving a service car event, checks if Q is empty. If it is, the server goes into idle mode. If Q is not empty, the server puts itself in a busy mode, removes one car C_i from the head of the queue Q, and charges its battery. In addition, the server calculates how long it will take to charge the car's battery as $(K_i - P_i)/M$, where M is the maximum rate at which power can be charged. The server then schedules another service car event to be sent to itself after this amount of time has elapsed.

As this simulation process runs, the program records interesting facts about the system, such as the number of cars in Q as time progresses and the fraction of time that the server is in idle versus busy modes. The maximum size of Q over the simulation's duration, and the average percentage of time the server is busy, are two examples of performance metrics that we might be interested in.

Some initiation is required for the simulation; specifically we need to know the values of D (which governs car inter-creation intervals), F (which governs battery size), and M (which is the maximum charging rate (power flow rate)). Thus D, F, and M are system parameters of the simulation. The values of these parameters must be set to reasonable values for the simulation to produce interpretable results. The choice of values requires detailed domain expertise—in this case, perhaps through consultation with experts, or by statistical analysis of real trace data from a real-world charging station.

Once these event-driven behaviors are defined, they can be translated into code. This is easy with an appropriate library that has subroutines for the creation, scheduling, and proper timing of events, queue manipulations, random variable generation, and collecting statistics. This charging station system will be implemented as a case study in Chapter 4, using a discrete-event simulation framework that will be designed and implemented in Chapter 2.

The performance measures (maximum size of Q over the simulation's duration, and the average percentage of time the server is busy) depend heavily on random choices made in the course of the simulation. Thus, the system is non-deterministic and the performance measures are themselves random variables. The distributions of these random variables using techniques such as Monte Carlo simulation will be discussed in Chapter 4.

1.3 Performance Evaluation Techniques

Modeling deals with the representation of interconnected subsystems and subprocesses with the eventual objective of obtaining an estimate of an aggregate systemic property, or *performance measure*. The mechanism by which one can obtain this estimate (1) efficiently and (2) with confidence is the subject of *performance evaluation*. In this section, we will lay out the methodological issues that lie therein.

A *performance evaluation technique* (PET) is a set of assumptions and analytical processes (applied in the context of a simulation), whose purpose is the efficient estimation of some performance measure. PETs fall into two broad classes: (1) direct measurement and (2) modeling:

1. **Direct measurement:** The most obvious technique to evaluate the performance of a system. However, limitations exist in the available traffic measurements because of the following reasons:
   (a) The direct measurement of a system is only available with operational systems. Direct measurement is not feasible with systems under design and development.
   (b) Direct measurement of the system may affect the measured system while obtaining the required data. This may lead to false measurements of the measured system.
   (c) It may not be practical to measure directly the level of end-to-end performance sustained on all paths of the system. Hence, to obtain the performance objectives, analytical models have to be set up to convert the raw measurements into meaningful performance measures.
2. **Modeling:** During the design and development phases of a system, modeling can be used to estimate the performance measures to be obtained when the system is

implemented. Modeling can be used to evaluate the performance of a working system, especially after the system undergoes some modifications. Modeling does not affect the measured system as it does in the direct measurement technique and it can be used during the design phases. However, modeling suffers from the following problems:

(a) The problem of *system abstraction*. This may lead to analyzing a model that does not represent the real system under evaluation.
(b) Problems in representing the workload of the system.
(c) Problems in obtaining performance measures for the model and mapping the results back to the real system.

Needless to say, the direct measurement of physical systems is *not* the subject of this book. Even within the modeling, only a sub-branch is considered here. To see where this sub-branch lies, it should be noted that at the heart of every model is its formal description. This can almost always be given in terms of a collection of interconnected queues. Once a system has been modeled as a collection of interconnected queues, there are two broadly defined approaches for determining actual performance measures: (1) analytical modeling and (2) simulation modeling. Analytical modeling of such a system seeks to deduce the parameters through mathematical derivation, and so leads into the domain of *queuing theory*—a fascinating mathematical subject for which many texts are already available. In contrast, simulation modeling (which is the subject of this book) determines performance measures by making *direct measurements of a simplified virtual system that has been created through computational means.*

In simulation modeling, there are few PETs that can be applied generally, though many PETs are based on modifications of the so-called Monte Carlo method that will be discussed in depth in Chapter 4. More frequently, each PET typically has to be designed on a case-by-case basis given the system at hand. The process of defining a PET involves:

1. Assumptions or simplifications of the properties of the atomic elements within a system and the logic underlying their evolution through interactions.[2]
2. Statistical assumptions on the properties of non-deterministic aspects of the system, e.g., its constituent waveforms.
3. Statistical techniques for aggregating performance measure estimates obtained from disparate simulations of different system configurations.[3]

[2] Note that this implies that a PET may be intertwined with the process of model construction, and because of this, certain PETs imply or require departure from an established model.
[3] At the simplest level, we can consider Monte Carlo simulation to be a PET. It satisfies our efficiency objective to the extent that the computations associated with missing or simplified operations reduce the overall computational burden.

To illustrate, let us consider an example of a PET methodology that might be applied in the study of a high data rate system over slowly fading channels. Without going into too many details, such a system's performance depends upon two random processes with widely different rates of change in time. One of the processes is "fast" in terms of its dynamism while the other is "slow." For example, the fast process in this case might be thermal noise, while the slow process might be signal fading. If the latter process is sufficiently slow relative to the first, then it can be approximated as being considered fixed with respect to the fast process. In such a system, the evolution of a received waveform could be simulated, by considering a sequence of time segments over each of which the state of the slow process is different, but fixed. We might simulate such a sequence of segments, and view each segment as being an experiment (conditioned on the state of the slow process) and obtain conditional performance metrics. The final performance measure estimate might then be taken as the average over these conditional estimates. This narrative illustrates the process of selecting and specifying a PET.

1.3.1 Example of Electric Car Battery Charging Station

In Section 1.2.1, a model of a charging station was described. What might the associated performance measures be when we simulate the model? There are many possible interesting aggregate performance measures we could consider, such as the number of cars in Q as time progresses and the fraction of time when the server is in idle versus busy modes.

1.3.1.1 Practical Considerations

Simulation is, in some sense, an act of pretending that one is dealing with a real object when actually one is working with an imitation of reality. It may be worthwhile to consider when such pretense is warranted, and the benefits/dangers of engaging in it.

1.3.1.2 Why Use Models?

Models reduce cost using different simplifying assumptions. A spacecraft simulator on a certain computer (or simulator system) is also a computer model of some aspects of the spacecraft. It shows on the computer screen the controls and functions that the spacecraft user is supposed to see when using the spacecraft. To train individuals using a simulator is more convenient, safer, and cheaper than a real spacecraft that usually cost billions of dollars. It is prohibitive (and impossible) to use a real spacecraft as a training system. Therefore, industry, commerce, and the military often use models rather than real experiments. Real experiments can be as costly and dangerous as real systems so, provided that models are adequate descriptions of reality, they can give results that are closer to those achieved by a real system analysis.

1.3.1.3 When to Use Simulations?

Simulation is used in many situations, such as:

1. When the analytical model/solution is not possible or feasible. In such cases, experts resort to simulations.
2. Many times, simulation results are used to verify analytical solutions in order to make sure that the system is modeled correctly using analytical approaches.
3. *Dynamic systems*, which involve randomness and change of state with time. An example is our electric car charging station where cars come and go unpredictably to charge their batteries. In such systems, it is difficult (sometimes impossible) to predict exactly what time the next car should arrive at the station.
4. *Complex dynamic systems*, which are so complex that when analyzed theoretically will require too many simplifications. In such cases, it is not possible to study the system and analyze it analytically. Therefore, simulation is the best approach to study the behavior of such a complex system.

1.3.1.4 How to Simulate?

Suppose one is interested in studying the performance of an electric car charging station (the example treated above). The behavior of this system may be described graphically by plotting the number of cars in the charging station and the state of the system. Every time a car arrives, the graph increases by one unit, while a departing car causes the graph to drop one unit. This graph, also called a *sample path*, could be obtained from observation of a real electric car charging station, but could also be constructed artificially. Such artificial construction and the analysis of the resulting sample path (or more sample paths in more complex cases) constitute the simulation process.

1.3.1.5 How Is Simulation Performed?

Simulations may be performed manually (on paper for instance). More frequently, however, the system model is written either as a computer program or as some kind of input to simulation software, as shown in Figure 1.1.

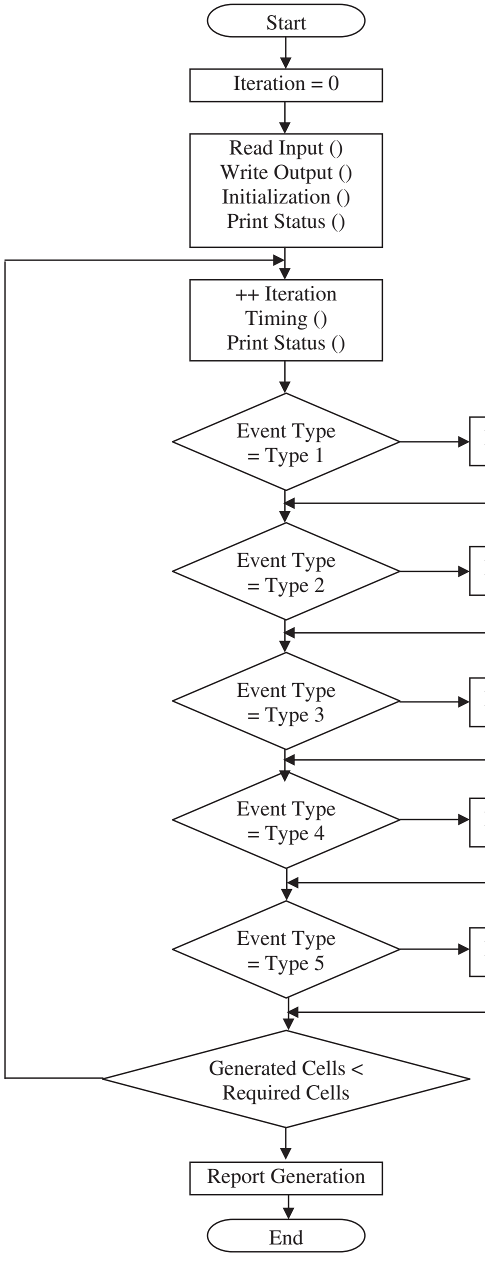

Figure 1.1 Overall structure of a simulation program

1.3.2 Common Pitfalls

In the following, we discuss the most common modeling and programming mistakes that lead to inaccurate or misleading simulation results:

1. **Inappropriate level of detail:** Normally, analytical modeling is carried out after some simplifications are adopted, since, without such simplifications, analysis tends to be intractable. In contrast, in simulation modeling, it is often tempting to include a very high level of simulation detail, since the approach is computational and not analytical. Such a decision is not always recommended, however, since much more time is needed to develop and run the computations. Even more importantly, having a model that is too detailed introduces a large number of interdependent parameters, whose influence on system performance becomes difficult to determine and isolate. For example, suppose we made our electric car charging station model include whether or not the server had a conversation with a passenger in the car, changed one of the car tires, and/or washed the car. In addition, each car had state variables concerning its battery charging level, passenger's gender, and battery type/size. If this model is used, it should contain details of this dynamic system. In this case, the simulation will get more complicated since it should contain details of the model that will make it so precise. But is this the best way to deal with the system? Is it worth including these details in the simulation model? Will the simulation results be better keeping the modeling details or it is better to leave them out since they will not add any necessary improvements to the results? Even if they do, will that be worth the added complexity to the system?
2. **Improper selection of the programming language:** The choice of programming language to be used in implementing the simulation model greatly affects the development time of the model. Special-purpose simulation languages require less time for development, while general-purpose languages are more efficient during the run time of the simulation program.
3. **Unverified models:** Simulation models are generally very large computer programs. These programs can contain several programming (or, even worse, logical) errors. If these programs are not verified to be correct, they will lead to misleading results and conclusions and invalid performance evaluation of the system.
4. **Invalid models:** Simulation programs may have no errors, but they may not represent the behavior of the real system that we want to evaluate. Hence, domain experts must verify the simulation assumptions codified by the mathematical model.
5. **Improperly selected initial conditions:** The initial conditions and parameter values used by a simulation program normally do not reflect the system's behavior in the steady state. Poorly selected initial conditions can lead to convergence to

atypical steady states, suggesting misleading conclusions about the system. Thus, domain experts must verify the initial conditions and parameters before they are used within the simulation program.

6. **Short run times:** System analysts may try to save time by running simulation programs for short periods. Short runs lead to false results that are strongly dependent on the initial conditions and thus do not reflect the true performance of the system in typical steady-state conditions.
7. **Poor random number generators:** Random number generators employed in simulation programs can greatly affect simulation results. It is important to use a random number generator that has been extensively tested and analyzed. The seeds that are supplied to the random number generator should also be selected to ensure that there is no correlation between the different processes in the system.
8. **Inadequate time estimate:** Most simulation models fail to give an adequate estimate of time needed for the development and implementation of the simulation model. The development, implementation, and testing of a simulation model require a great deal of time and effort.
9. **No achievable goals:** Simulation models may fail because no specific, achievable goals are set before beginning the process of developing the simulation model.
10. **Incomplete mix of essential skills:** For a simulation project to be successful, it should employ personnel who have different types of skills, such as project leaders and programmers, people who have skills in statistics and modeling, and people who have a good domain knowledge and experience with the actual system being modeled.
11. **Inadequate level of user participation:** Simulation models that are developed without end user's participation are usually not successful. Regular meetings among end users, system developers, and system implementers are important for a successful simulation program.
12. **Inability to manage the simulation project:** Most simulation projects are extremely complex, so it is important to employ software engineering tools to keep track of the progress and functionality of a project.

1.3.3 Types of Simulation Techniques

The most important types of simulations described in the literature that are of special importance to engineers are:

1. **Emulation:** The process of designing and building hardware or firmware (i.e., prototype) that imitates the functionality of the real system.
2. **Monte Carlo simulation:** Any simulation that has no time axis. Monte Carlo simulation is used to model probabilistic phenomena that do not change with time,

or to evaluate non-probabilistic expressions using probabilistic techniques. This kind of simulation will be discussed in greater detail in Chapter 4.

3. **Trace-driven simulation:** Any simulation that uses an ordered list of real-world events as input.
4. **Continuous-event simulation:** In some systems, the state changes occur all the time, not merely at discrete times. For example, the water level in a reservoir with given in- and outflows may change all the time. In such cases "continuous simulation" is more appropriate, although discrete-event simulation can serve as an approximation.
5. **Discrete-event simulation:** A discrete-event simulation is characterized by two features: (1) within any interval of time, one can find a subinterval in which no event occurs and no state variables change; (2) the number of events is finite. All discrete-event simulations have the following components:
   (a) *Event queue*: A list that contains all the events waiting to happen (in the future). The implementation of the event list and the functions to be performed on it can significantly affect the efficiency of the simulation program.
   (b) *Simulation clock*: A global variable that represents the simulated time. Simulation time can be advanced by *time-driven* or *event-driven* methods. In the time-driven approach, time is divided into constant, small increments, and then events occurring within each increment are checked. In the event-driven approach, on the other hand, time is incremented to the time of the next imminent event. This event is processed and then the simulation clock is incremented again to the time of the next imminent event, and so on. This latter approach is the one that is generally used in computer simulations.
   (c) *State variables*: Variables that together completely describe the state of the system.
   (d) *Event routines*: Routines that handle the occurrence of events. If an event occurs, its corresponding event routine is executed to update the state variables and the event queue appropriately.
   (e) *Input routine*: The routine that gets the input parameters from the user and supplies them to the model.
   (f) *Report generation routine*: The routine responsible for calculating results and printing them out to the end user.
   (g) *Initialization routine*: The routine responsible for initializing the values of the various state variables, global variables, and statistical variables at the beginning of the simulation program.
   (h) *Main program*: The program where the other routines are called. The main program calls the initialization routine; the input routine executes various iterations, finally calls the report generation routine, and terminates the simulation. Figure 1.1 shows the overall structure that should be followed to implement a simulation program.

1.4 Development of Systems Simulation

Discrete-event systems are dynamic systems that evolve in time by the occurrence of events at possibly irregular time intervals. Examples include traffic systems, flexible manufacturing systems, computer communications systems, production lines, coherent lifetime systems, and flow networks. Most of these systems can be modeled in terms of discrete events whose occurrence causes the system to change state. In designing, analyzing, and simulating such complex systems, one is interested not only in performance evaluation, but also in analysis of the sensitivity of the system to design parameters and optimal selection of parameter values.

A typical stochastic system has a large number of control parameters, each of which can have a significant impact on the performance of the system. An overarching objective of simulation is to determine the relationship between system behavior and input parameter values, and to estimate the relative importance of these parameters and their relationships to one another as mediated by the system itself. The technique by which this information is deduced is termed *sensitivity analysis*. The methodology is to apply small perturbations to the nominal values of input parameters and observe the effects on system performance measures. For systems simulation, variations of the input parameter values cannot be made infinitely small (since this would then require an infinite number of simulations to be conducted). The *sensitivity* (of the performance measure with respect to an input parameter) is taken to be an approximation of the partial derivative of the performance measure with respect to the parameter value.

The development process of system simulation involves some or all of the following stages:

- **Problem formulation:** This involves identifying the controllable and uncontrollable inputs (see Figure 1.2), identifying constraints on the decision variables, defining a measure of performance (i.e., an objective function), and developing a preliminary model structure to interrelate the system inputs to the values of the performance measure.

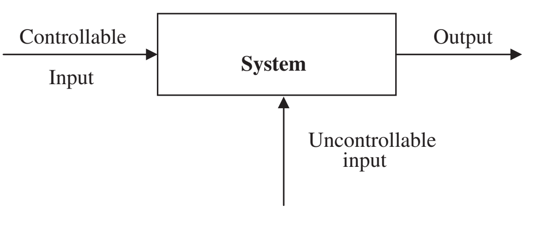

Figure 1.2 System block diagram

- **Data collection and analysis:** Decide *what* data to collect about the real system, and *how much* to collect. This decision is a tradeoff between cost and accuracy.
- **Simulation model development:** Acquire sufficient understanding of the system to develop an appropriate conceptual, logical model of the entities and their states, as well as the events codifying the interactions between entities (and time). This is the heart of simulation design.
- **Model validation, verification, and calibration:** In general, verification focuses on the internal consistency of a model, while validation is concerned with the correspondence between the model and the reality. The term *validation* is applied to those processes that seek to determine whether or not a simulation is correct with respect to the "real" system. More precisely, validation is concerned with the question "are we building the right system?" Verification, on the other hand, seeks to answer the question "are we building the system right?" Verification checks that the implementation of the simulation model (program) corresponds to the model. Validation checks that the model corresponds to reality. Finally, calibration checks that the data generated by the simulation matches real (observed) data.
- **Validation:** The process of comparing the model's output to the behavior of the phenomenon. In other words, comparing model execution to reality (physical or otherwise). This process and its role in simulation are described in Figure 1.3.
- **Verification:** The process of comparing the computer code to the model to ensure that the code is a correct implementation of the model.
- **Calibration:** The process of parameter estimation for a model. Calibration is a tweaking/tuning of existing parameters which usually does not involve the introduction of new ones, changing the model structure. In the context of optimization,

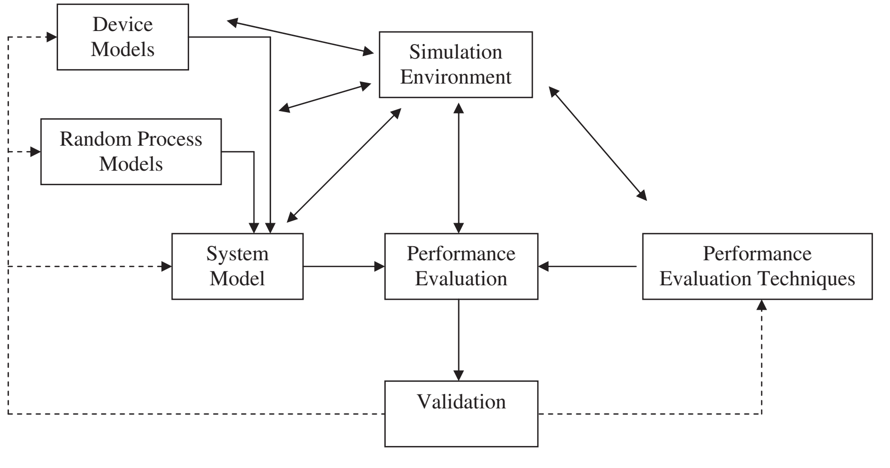

Figure 1.3 Validation process

calibration is an optimization procedure involved in system identification or during the experimental design.

- **Input and output analysis:** Discrete-event simulation models typically have stochastic components that mimic the probabilistic nature of the system under consideration. Successful input modeling requires a close match between the input model and the true underlying probabilistic mechanism associated with the system. The input data analysis is to model an **element** (e.g., arrival process, service times) in a discrete-event simulation given a data set collected on the element of interest. This stage performs intensive error checking on the input data, including external, policy, random, and deterministic variables. System simulation experiments aim to learn about its behavior. Careful planning, or designing, of simulation experiments is generally a great help, saving time and effort by providing efficient ways to estimate the effects of changes in the model's inputs on its outputs. Statistical experimental design methods are mostly used in the context of simulation experiments.
- **Performance evaluation and "what-if" analysis:** The "what-if" analysis involves trying to run the simulation with different input parameters (i.e., under different "scenarios") to see how performance measures are affected.
- **Sensitivity estimation:** Users must be provided with affordable techniques for sensitivity analysis if they are to understand the relationships and tradeoffs between system parameters and performance measures in a meaningful way that allows them to make good system design decisions.
- **Optimization:** Traditional optimization techniques require gradient estimation. As with sensitivity analysis, the current approach for optimization requires intensive simulation to construct an approximate surface response function. Sophisticated simulations incorporate gradient estimation techniques into convergent algorithms such as Robbins–Monroe in order to efficiently determine system parameter values which optimize performance measures. There are many other gradient estimation (sensitivity analysis) techniques, including: **l**ocal information, structural properties, response surface generation, the goal-seeking problem, optimization, the "what-if" problem, and meta-modeling.
- **Report generating:** Report generation is a critical link in the communication process between the model and the end user.

Figure 1.4 shows a block diagram describing the development process for systems simulation. It describes each stage of the development as above.

1.4.1 Overview of a Modeling Project Life Cycle

A *software life cycle model* (SLCM) is a schematic of the major components of software development work and their interrelationships in a graphical framework

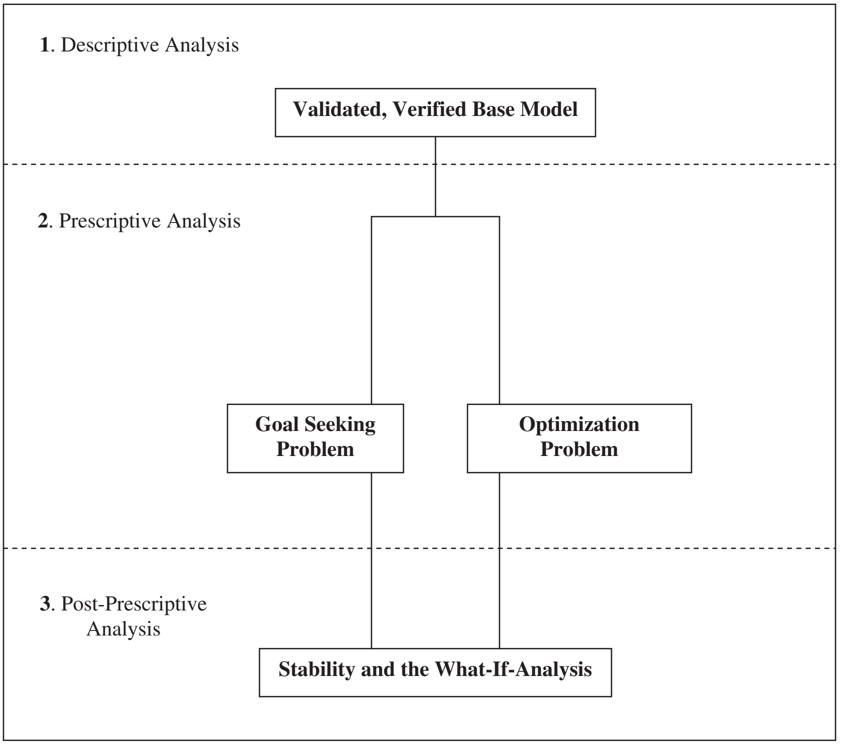

Figure 1.4 Development process for simulation

that can be easily understood and communicated. The SLCM partitions the work to be done into manageable work units. Having a defined SLCM for your project allows you to:

1. Define the work to be performed.
2. Divide up the work into manageable pieces.
3. Determine project milestones at which project performance can be evaluated.
4. Define the sequence of work units.
5. Provide a framework for definition and storage of the deliverables produced during the project.
6. Communicate your development strategy to project stakeholders.

An SLCM achieves this by:

1. Providing a simple graphical representation of the work to be performed.

2. Allowing focus on important features of the work, downplaying excessive detail.
3. Providing a standard work unit hierarchy for progressive decomposition of the work into manageable chunks.
4. Providing for changes (tailoring) at low cost.

Before specifying how this is achieved, we need to classify the life cycle processes dealing with the software development process.

1.4.2 Classifying Life Cycle Processes

Figure 1.5 shows the three main classes of a software development process and gives examples of the members of each class:

- **Project support processes:** are involved with the management of the software development exercise. They are performed throughout the life of the project.
- **Development processes:** embody all work that directly contributes to the development of the project deliverable. They are typically interdependent.
- **Integral processes:** are common processes that are performed in the context of more than one development activity. For example, the review process is performed in the context of requirements definition, design, and coding.

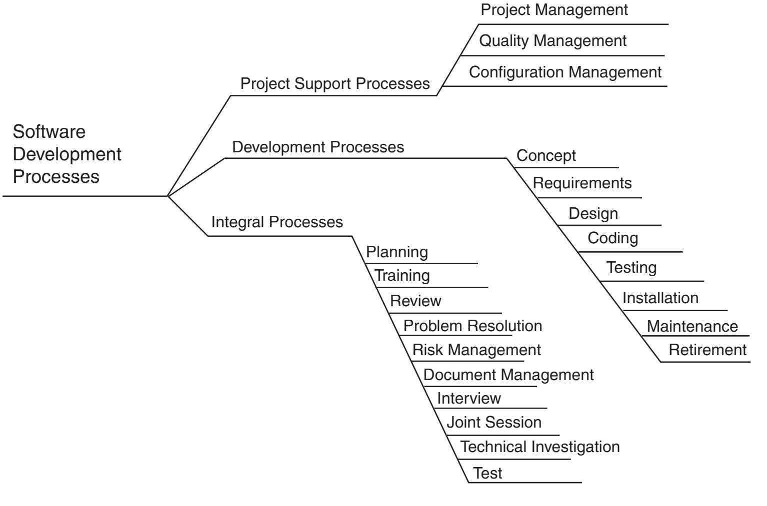

Figure 1.5 Software life cycle process classifications

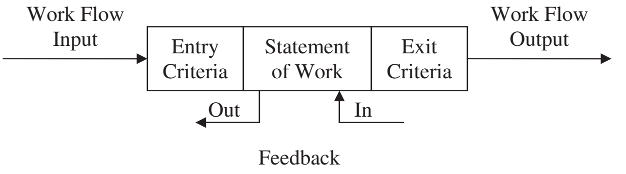

Figure 1.6 Components of a work unit

1.4.3 Describing a Process

Processes are described in terms of a series of work units. Work units are logically related chunks of work. For example, all preliminary design effort is naturally chunked together. Figure 1.6 describes the components of a work unit.

A work unit is described in terms of:

- **Work flow input/output:** Work flows are the work products that flow from one work unit to the next. For example, in Figure 1.6, the design specification flows from the output of the design work unit to the input of the code work unit. Work flows are the deliverables from the work unit. All work units must have a deliverable. The life cycle model should provide detailed descriptions of the format and content of all deliverables.
- **Entry criteria:** The conditions that must exist before a work unit can commence.
- **Statement of work (SOW):** The SOW describes the work to be performed on the work flow inputs to create the outputs.
- **Exit criteria:** The conditions that must exist for the work to be deemed complete.

The above terms are further explained through the usage of examples in Table 1.2.

Feedback paths are the paths by which work performed in one work unit impacts work either in progress or completed in a preceding work unit. For example, the model depicted in Figure 1.7 allows for the common situation where the act of coding often uncovers inconsistencies and omissions in the design. The issues raised by programmers then require a reworking of the baseline design document.

Defining feedback paths provides a mechanism for iterative development of work products. That is, it allows for the real-world fact that specifications and code are seldom complete and correct at their first approval point. The feedback path allows for planning, quantification, and control of the rework effort. Implementing feedback paths on a project requires the following procedures:

1. A procedure to raise issues or defects with baseline deliverables.
2. A procedure to review issues and approve rework of baseline deliverables.
3. Allocation of budget for rework in each project phase.

Table 1.2 Examples of entry and exit criteria and statements of work

Entry criteria	Statement of work	Exit criteria
1. Prior tasks complete	1. Produce deliverable	1. Deliverable complete
2. Prior deliverables approved and baselined	2. Interview user	2. Deliverable approved
3. Tasks defined for this work unit	3. Conduct review	3. Test passed
4. Deliverables defined for this work unit	4. Conduct test	4. Acceptance criteria satisfied
5. Resources available	5. Conduct technical investigation	5. Milestone reached
6. Responsibilities defined	6. Perform rework	
7. Procedures defined		
8. Process measurements defined		
9. Work authorized		

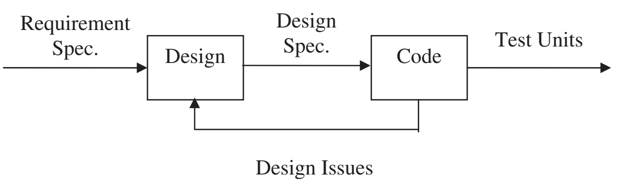

Figure 1.7 Example of a feedback path

1.4.4 Sequencing Work Units

Figure 1.8 provides an example of a life cycle model constructed by sequencing work units. We see work flows of *requirements specification* and *design specification.* The work units are design, code, and test, and the feedback paths carry requirements, design, and coding issues.

Note that the arrows in a life cycle model do not signify precedence. For example, in Figure 1.8, the design does not have to be complete for coding to commence.

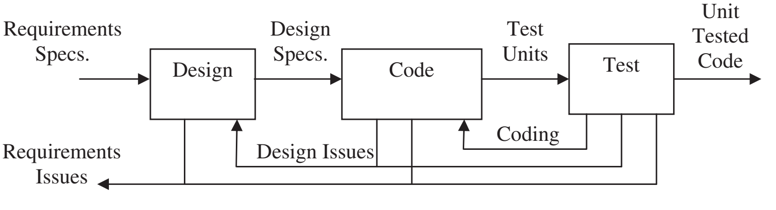

Figure 1.8 Life cycle model

Design may be continuing while some unrelated elements of the system are being coded.

1.4.5 Phases, Activities, and Tasks

A primary purpose of the life cycle model is to communicate the work to be done among human beings. Conventional wisdom dictates that, to guarantee comprehension, a single life cycle model should therefore not have more than nine work units. Clearly this would not be enough to describe even the simplest projects.

In large-scale projects, the solution is to decompose large work units into a set of levels with each level providing more detail about the level above it. Figure 1.9 depicts three levels of decomposition:

1. **The phase level:** Phases describe the highest levels of activity in the project. Examples of phases might be requirements capture and design. Phases are typically used in the description of development processes.
2. **The activity level:** Activities are logically related work units within a phase. An activity is typically worked on by a team of individuals. Examples of activities might include interviewing users as an activity within the requirements capture phase.
3. **The task level:** Tasks are components of an activity that are typically performed by one or two people. For example, conducting a purchasing manager interview is a specific task that is within the interviewing users activity. Tasks are where the work

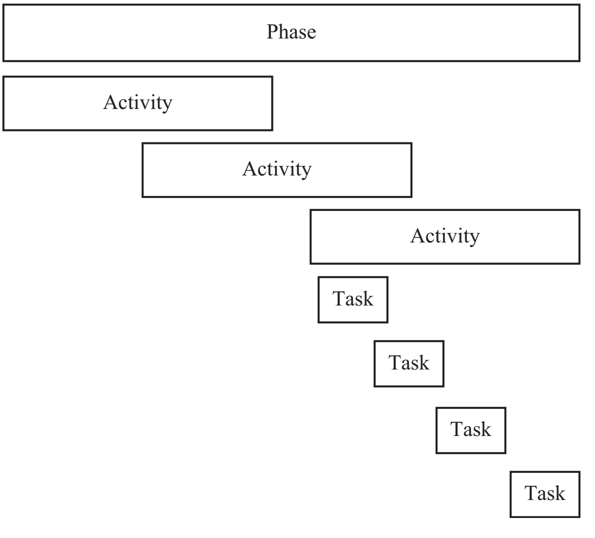

Figure 1.9 Work unit decomposition

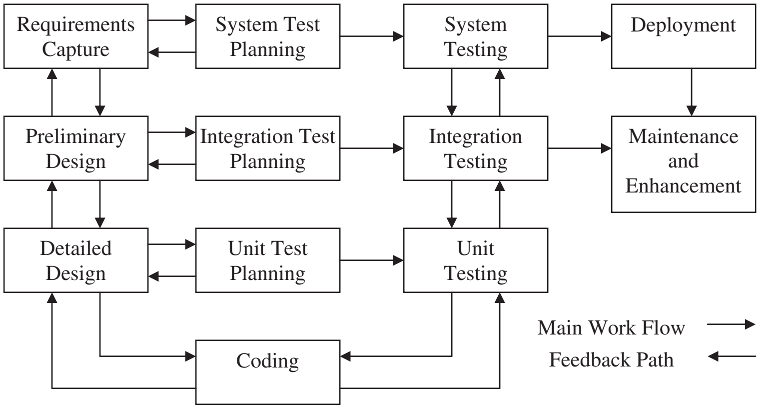

Figure 1.10 A generic project model for software development

is done. A task will have time booked to it on a time sheet. Conventional guidelines specify that a single task should be completed in an average elapsed time of 5 working days and its duration should not exceed 15 working days.

Note that phases, activities, and tasks are all types of a work unit. You can therefore describe them all in terms of entry criteria, statement of work, and exit criteria.

As a concrete example, Figure 1.10 provides a phase-level life cycle model for a typical software development project.

1.5 Summary

In this chapter, we introduced the foundations of modeling and simulation. We highlighted the importance, the needs, and the usage of simulation and modeling in practical and physical systems. We also discussed the different approaches to modeling that are used in practice and discussed at a high level the methodology that should be followed when executing a modeling project.

Recommended Reading

[1] J. Sokolowski and C. Banks (Editors), *Principles of Modeling and Simulation: A Multidisciplinary Approach*, John Wiley & Sons, Inc., 2009.
[2] J. Banks (Editor), *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, John Wiley & Sons, Inc., 1998.
[3] B. Zeigler, H. Praehofer, and T.-G. Kim, *Theory of Modeling and Simulation*, 2nd Edition, Academic Press, 2000.
[4] A. Law, *Simulation Modeling and Analysis*, 4th Edition, McGraw-Hill, 2007.