CHAPTER 1

What Is Simulation?

Simulation refers to a broad collection of methods and applications to mimic the behavior of real systems, usually on a computer with appropriate software. In fact, "simulation" can be an extremely general term since the idea applies across many fields, industries, and applications. These days, simulation is more popular and powerful than ever since computers and software are better than ever.

This book gives you a comprehensive treatment of simulation in general and the Arena simulation software in particular. We cover the general idea of simulation and its logic in Chapters 1 and 2 (including a bit about using spreadsheets to simulate) and Arena in Chapters 3–9. We don't, however, intend for this book to be a complete reference on everything in Arena (that's what the help systems in the software are for). In Chapter 10, we show you how to integrate Arena with external files and other applications and give an overview of some advanced Arena capabilities. In Chapter 11, we introduce you to continuous and combined discrete/continuous modeling with Arena. Chapters 12-13 cover issues related to planning and interpreting the results of simulation experiments, as well as managing a simulation project. Appendix A is a detailed account of a simulation project carried out for *The Washington Post* newspaper. Appendix B provides a quick review of probability and statistics necessary for simulation. Appendix C describes Arena's probability distributions, and Appendix D provides software installation instructions. After reading this book, you should be able to model systems with Arena and carry out effective and successful simulation studies.

This chapter touches on the general notion of simulation. In Section 1.1, we describe some general ideas about how you might study models of systems and give some examples of where simulation has been useful. Section 1.2 contains more specific information about simulation and its popularity, mentions some good things (and one bad thing) about simulation, and attempts to classify the many different kinds of simulations that people do. In Section 1.3, we talk a little bit about software options. Finally, Section 1.4 traces changes over time in how and when simulation is used. After reading this chapter, you should have an appreciation for where simulation fits in, the kinds of things it can do, and how Arena might be able to help you do them.

1.1 Modeling

Simulation, like most analysis methods, involves systems and models of them. So in this section, we give you some examples of models and describe options for studying them to learn about the corresponding system.

1.1.1 What's Being Modeled?

Computer simulation deals with models of systems. A *system* is a facility or process, either actual or planned, such as:

- A manufacturing plant with machines, people, transport devices, conveyor belts, and storage space.
- A bank with different kinds of customers, servers, and facilities like teller windows, automated teller machines (ATMs), loan desks, and safety deposit boxes.
- An airport with departing passengers checking in, going through security, going to the departure gate, and boarding; departing flights contending for push-back tugs and runway slots; arriving flights contending for runways, gates, and arrival crew; arriving passengers moving to baggage claim and waiting for their bags; and the baggage-handling system dealing with delays, security issues, and equipment failure.
- A distribution network of plants, warehouses, and transportation links.
- An emergency facility in a hospital, including personnel, rooms, equipment, supplies, and patient transport.
- A field-service operation for appliances or office equipment, with potential customers scattered across a geographic area, service technicians with different qualifications, trucks with different parts and tools, and a central depot and dispatch center.
- A computer network with servers, clients, disk drives, tape drives, printers, networking capabilities, and operators.
- A freeway system of road segments, interchanges, controls, and traffic.
- A central insurance claims office where a lot of paperwork is received, reviewed, copied, filed, and mailed by people and machines.
- A criminal-justice system of courts, judges, support staff, probation officers, parole agents, defendants, plaintiffs, convicted offenders, and schedules.
- A chemical-products plant with storage tanks, pipelines, reactor vessels, and railway tanker cars in which to ship the finished product.
- A fast-food restaurant with different types of staff, customers, and equipment.
- A supermarket with inventory control, checkout, and customer service.
- A theme park with rides, stores, restaurants, workers, guests, and parking lots.
- The response of emergency personnel to the occurrence of a catastrophic event.
- A network of shipping ports including ships, containers, cranes, and landside transport.
- A military operation including supplies, logistics, and combat engagement.

People often study a system to measure its performance, improve its operation, or design it if it doesn't exist. Managers or controllers of a system might also like to have a readily available aid for day-to-day operations, like help in deciding what to do in a factory if an important machine goes down.

We're even aware of managers who requested that simulations be constructed but didn't really care about the final results. Their primary goal was to focus attention on understanding how their system worked. Often simulation analysts find that the process of

defining how the system works, which must be done before you can start developing the simulation model, provides great insight into what changes need to be made. Part of this is due to the fact that rarely is there one individual responsible for understanding how an entire system works. There are experts in machine design, material handling, processes, and so on, but not in the day-to-day operation of the system. So as you read on, be aware that simulation is much more than just building a model and conducting a statistical experiment. There is much to be learned at each step of a simulation project, and the decisions you make along the way can greatly affect the significance of your findings.

1.1.2 How About Just Playing with the System?

It might be possible to experiment with the actual physical system. For instance:

- Some cities have installed entrance-ramp traffic lights on their freeway systems to experiment with different sequencing to find settings that make rush hour as smooth and safe as possible.
- A supermarket manager might try different policies for inventory control and checkout-personnel assignment to see what combinations seem to be most profitable and provide the best service.
- An airline could test the expanded use of automated check-in kiosks (and employees to urge passengers to use them) to see if this speeds check-in.
- A computer facility can experiment with different network layouts and job priorities to see how they affect machine utilization and turnaround.

This approach certainly has its advantages. If you can experiment directly with the system and know that nothing else about it will change significantly, then you're unquestionably looking at the right thing and needn't worry about whether a model or proxy for the system faithfully mimics it for your purposes.

1.1.3 Sometimes You Can't (or Shouldn't) Play with the System

In many cases, it's just too difficult, costly, or downright impossible to do physical studies on the system itself.

- Obviously, you can't experiment with alternative layouts of a factory if it's not yet built.
- Even in an existing factory, it might be very costly to change to an experimental layout that might not work out anyway.
- It would be hard to run twice as many customers through a bank to see the effect of closing a nearby branch.
- Trying a new check-in procedure at an airport might initially cause a lot of people to miss their flights if there are unforeseen problems with the new procedure.
- Fiddling around with emergency room staffing in a hospital clearly won't do.

In these situations, you might build a *model* to serve as a stand-in for studying the system and ask pertinent questions about what *would* happen in the system *if* you did this or that, or *if* some situation beyond your control were to develop. *Nobody gets hurt, and your freedom to try wide-ranging ideas with the model could uncover attractive alternatives that you might not have been able to try with the real system.*

However, you have to build models carefully and with enough detail so that what you learn about the model will never[1] be different from what you would have learned about the system by playing with it directly. This is called model *validity*, and we'll have more to say about it later, in Chapter 13.

Types Of models

1.1.4 Physical Models

There are lots of different kinds of models. Maybe the first thing the word evokes is a physical replica or scale model of the system, sometimes called an *iconic* model. For instance:

- People have built *tabletop* models of material handling systems that are miniature versions of the facility, not unlike electric train sets, to consider the effect on performance of alternative layouts, vehicle routes, and transport equipment.
- A full-scale version of a fast-food restaurant placed inside a warehouse to experiment with different service procedures was described by Swart and Donno (1981). In fact, most large fast-food chains now have full-scale restaurants in their corporate office buildings for experimentation with new products and services.
- Simulated control rooms have been developed to train operators for nuclear power plants.
- Physical flight simulators are widely used to train pilots. There are also flight-simulation computer programs, with which you may be familiar in game form, that represent purely logical models executing inside a computer. Further, physical flight simulators might have computer screens to simulate airport approaches, so they have elements of both physical and computer-simulation models.

Although iconic models have proven useful in many areas, we won't consider them.

1.1.5 Logical (or Mathematical) Models

Instead, we'll consider *logical* (or *mathematical*) models of systems. Such a model is just a set of approximations and assumptions, both structural and quantitative, about the way the system does or will work.

A logical model is usually represented in a computer program that's exercised to address questions about the model's behavior; if your model is a valid representation of your system, you hope to learn about the system's behavior too. And since you're dealing with a mere computer program rather than the actual system, it's usually easy, cheap, and fast to get answers to a lot of questions about the model and system by simply manipulating the program's inputs and form. Thus, you can make your mistakes on the computer where they don't count, rather than for real where they do. As in many other fields, recent dramatic increases in computing power (and decreases in computing costs) have impressively advanced your ability to carry out computer analyses of logical models.

1.1.6 What Do You Do with a Logical Model?

After making the approximations and stating the assumptions for a valid logical model of the target system, you need to find a way to deal with the model and analyze its behavior.

[1] Well, hardly ever.

If the model is simple enough, you might be able to use traditional mathematical tools like queueing theory, differential-equation methods, or something like linear programming to get the answers you need. This is a nice situation since you might get fairly simple formulas to answer your questions, which can easily be evaluated numerically; working with the formula (for instance, taking partial derivatives of it with respect to controllable input parameters) might provide insight itself. Even if you don't get a simple closed-form formula, but rather an algorithm to generate numerical answers, you'll still have exact answers (up to roundoff, anyway) rather than estimates that are subject to uncertainty.

However, most systems that people model and study are pretty complicated, so that *valid* models[2] of them are pretty complicated too. For such models, there may not be exact mathematical solutions worked out, which is where simulation comes in.

1.2 Computer Simulation

Computer simulation refers to methods for studying a wide variety of models of real-world systems by numerical evaluation using software designed to imitate the system's operations or characteristics, often over time. From a practical viewpoint, simulation is the process of designing and creating a computerized model of a real or proposed system for the purpose of conducting numerical experiments to give us a better understanding of the behavior of that system for a given set of conditions. Although it can be used to study simple systems, the real power of this technique is fully realized when we use it to study complex systems.

While simulation may not be the only tool you could use to study the model, it's frequently the method of choice. The reason for this is that the simulation model can be allowed to become quite complex, if needed to represent the system faithfully, and you can still do a simulation analysis. Other methods may require stronger simplifying assumptions about the system to enable an analysis, which might bring the validity of the model into question.

1.2.1 Popularity and Advantages

Over the last two or three decades, simulation has been consistently reported as the most popular operations research tool:

- Rasmussen and George (1978) asked M.S. graduates from the Operations Research Department at Case Western Reserve University (of which there are many since that department was founded a long time ago) about the value of methods after graduation. The first four methods were *statistical analysis*, *forecasting*, *systems analysis*, and *information systems*, all of which are very broad and general categories. Simulation was next, and ranked higher than other more traditional operations research tools like linear programming and queueing theory.

[2] You can always build a simple (maybe simplistic) model of a complicated system, but there's a good chance that it won't be valid. If you go ahead and analyze such a model, you may be getting nice, clean, simple answers to the wrong questions. This is sometimes called a Type III Error—working on the wrong problem (statisticians have already claimed Type I and Type II Errors).

- Thomas and DaCosta (1979) gave analysts in 137 large firms a list of tools and asked them to check off which ones they used. Statistical analysis came in first, with 93% of the firms reporting that they use it (it's hard to imagine a large firm that wouldn't), followed by simulation (84%). Again, simulation came in higher than tools like linear programming, PERT/CPM, inventory theory, and nonlinear programming.
- Shannon, Long, and Buckles (1980) surveyed members of the Operations Research Division of the American Institute of Industrial Engineers (now the Institute of Industrial Engineers) and found that among the tools listed, simulation ranked first in utility and interest. Simulation was second in familiarity, behind linear programming, which might suggest that simulation should be given stronger emphasis in academic curricula.
- Forgionne (1983); Harpell, Lane, and Mansour (1989); and Lane, Mansour, and Harpell (1993) all report that, in terms of utilization of methods by practitioners in large corporations, statistical analysis was first and simulation was second. Again, though, academic curricula seem to be behind since linear programming was more frequently *taught*, as opposed to being *used* by practitioners, than was simulation.
- Morgan (1989) reviewed many surveys of the above type and reported that "heavy" use of simulation was consistently found. Even in an industry with the lowest reported use of operations research tools (motor carriers), simulation ranked first in usage.

The main reason for simulation's popularity is its ability to deal with very complicated models of correspondingly complicated systems. This makes it a versatile and powerful tool. Another reason for simulation's increasing popularity is the obvious improvement in performance/price ratios of computer hardware, making it ever more cost effective to do what was prohibitively expensive computing just a few years ago. Finally, advances in simulation software power, flexibility, and ease of use have moved the approach from the realm of tedious and error-prone, low-level programming to the arena of quick and valid decision making.

Our guess is that simulation's popularity and effectiveness are now even greater than reported in the surveys described above, precisely due to these advances in computer hardware and software.

1.2.2 The Bad News

Shortcomings of simulation

However, simulation isn't *quite* paradise, either.

Because many real systems are affected by uncontrollable and random inputs, many simulation models involve random, or *stochastic*, input components, causing their output to be random too. For example, a model of a distribution center would have arrivals, departures, and lot sizes arising randomly according to particular probability distributions, which will propagate through the model's logic to cause output performance measures like throughput and cycle times to be random as well. So running a stochastic simulation once is like performing a random physical experiment once, or watching the distribution center for one day—you'll probably see something different

next time, even if you don't change anything yourself. In many simulations, as the time frame becomes longer (like months instead of a day), most results averaged over the run will tend to settle down and become less variable, but it can be hard to determine how long is "long enough" for this to happen. Moreover, the model or study might dictate that the simulation stop at a particular point (for instance, a bank is open from 9 to 5), so running it longer to calm the output is inappropriate.

Thus, you have to think carefully about designing and analyzing simulation experiments to take account of this uncertainty in the results, especially if the appropriate time frame for your model is relatively short. We'll return to this idea repeatedly in the book and illustrate proper statistical design and analysis tools, some of which are built into Arena, but others you have to worry about yourself.

Even though simulation output may be uncertain, we can deal with, quantify, and reduce this uncertainty. You might be able to get rid of the uncertainty completely by making a lot of over-simplifying assumptions about the system; this would get you a nice, simple model that will produce nice, non-random results. Unfortunately, though, such an over-simplified model will probably not be a *valid* representation of the system, and the error due to such model invalidity is impossible to measure or reliably reduce. For our money, we'd prefer an approximate answer to the right problem rather than an exact answer to the wrong problem (remember the Type III Error?).

1.2.3 Different Kinds of Simulations

There are a lot of ways to classify simulation models, but one useful way is along these three dimensions:

- **Static vs. Dynamic:** Time doesn't play a natural role in static models but does in dynamic models. The Buffon needle problem, described at the beginning of Section 1.3.1, is a static simulation. The small manufacturing model described in Chapters 2 and 3 is a dynamic model. Most operational models are dynamic; Arena was designed with them in mind, so our primary focus will be on such models (except for Section 2.7.1, which develops a static model).
- **Continuous vs. Discrete:** In a continuous model, the state of the system can change continuously over time; an example would be the level of a reservoir as water flows in and is let out, and as precipitation and evaporation occur. In a discrete model, though, change can occur only at separated points in time, such as a manufacturing system with parts arriving and leaving at specific times, machines going down and coming back up at specific times, and breaks for workers. You can have elements of both continuous and discrete change in the same model, which are called *mixed continuous-discrete models*; an example might be a refinery with continuously changing pressure inside vessels and discretely occurring shutdowns. Arena can handle continuous, discrete, and mixed models; our focus will be on discrete models for most of the book, though Chapter 11 discusses continuous and mixed models.
- **Deterministic vs. Stochastic:** Models that have no random input are deterministic; a strict appointment-book operation with fixed service times is an example. Stochastic models, on the other hand, operate with at least some inputs being

random—like a bank with randomly arriving customers requiring varying service times. A model can have both deterministic and random inputs in different components; which elements are modeled as deterministic and which as random are issues of modeling realism. Arena easily handles deterministic and stochastic inputs to models and provides many different probability distributions and processes that you can use to represent the random inputs. Since we feel that at least some element of uncertainty is usually present in reality, most of our illustrations involve random inputs somewhere in the model. As noted earlier, though, stochastic models produce uncertain output, which is a fact you must consider carefully in designing and interpreting the runs in your project.

1.3 How Simulations Get Done

If you've determined that a simulation of some sort is appropriate, you next have to decide how to carry it out. In this section, we discuss options for running a simulation, including software.

1.3.1 By Hand

In the beginning, people really *did* do simulations by hand (we'll show you just one, which is painful enough, in Section 2.4).

For instance, around 1733 a fellow by the name of Georges Louis Leclerc (who later was invited into the nobility, due no doubt to his simulation prowess, as Comte de Buffon) described an experiment to estimate the value of π. If you toss a needle of length

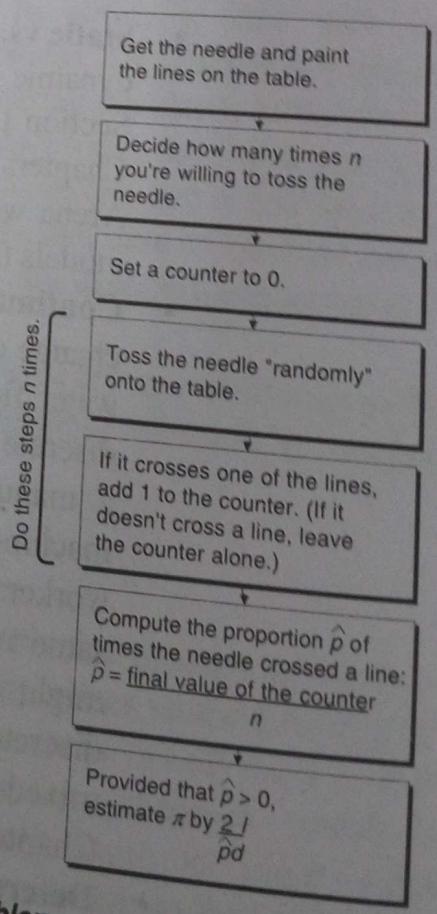


Figure 1-1. The Buffon Needle Problem

l onto a table painted with parallel lines spaced d apart (d must be $\geq l$), it turns out that the needle will cross a line with probability $p = 2l/(\pi d)$. So Figure 1-1 shows a simulation experiment to estimate the value of π. (Don't try this at home, or at least not with a big needle.)

Though this experiment may seem pretty simple (probably even silly) to you, there are some aspects of it that are common to most simulations:

- The purpose is to estimate something (in this case, π) whose value would be hard to compute exactly (okay, maybe in 1733 that was true).
- The estimate we get at the end is not going to be exactly right; that is, it has some error associated with it, and it might be nice to get an idea of how large that error is likely to be.
- It seems intuitive that the more tosses we make (that is, the bigger n is), the smaller the error is likely to be and thus the better the estimate is likely to be.
- In fact, you could do a *sequential* experiment and just keep tossing until the probable error is small enough for you to live with, instead of deciding on the number n of tosses beforehand.
- You might be able to reduce the error without increasing the number of tosses if you invest a little up-front work. Weld a second needle to the first so they cross at right angles at their midpoints; such a weapon is called a *Buffon cross*. Leave the lines on the table alone. On each toss, record separately whether each needle crosses a line (it could be that they both cross, neither crosses, or just one but not the other crosses), and get two different estimates of π. It's intuitive (and, happily, true) that whether one needle crosses a line is negatively correlated with whether the other one does, so the two estimates of π will be negatively correlated with each other. The average of the two estimates is also an unbiased estimate of π, but will have less variance than a single-needle estimate from the same number of tosses since it's likely that if one estimate is high, the other one will be low. (This is a physical analog of what's called a *variance-reduction technique*, specifically *antithetic variates*, and is discussed in Section 12.4.2. It seems like some kind of cheat or swindle, but it's really a fair game.)

We'll come back to these kinds of issues as we talk about more interesting and helpful simulations. (For more on the Buffon needle problem, as well as other such interesting historical curiosities, see Morgan, 1984.)

In the 1920s and 1930s, statisticians began using random-number machines and tables in numerical experiments to help them develop and understand statistical theory. For instance, Walter A. Shewhart (the quality control pioneer) did numerical experiments by drawing numbered chips from a bowl to study the first control charts. Guinness Brewery employee W. S. Gossett did similar numerical sampling experiments to help him gain insight into what was going on in mathematical statistics. (To protect his job at Guinness, he published his research under the pseudonym "Student" and also developed the t distribution used widely in statistical inference.) Engineers, physicists, and mathematicians have used various kinds of hand-simulation ideas for many years on a wide variety of problems.

1.3.2 Programming in General-Purpose Languages

As digital computers appeared in the 1950s and 1960s, people began writing computer programs in general-purpose procedural languages like FORTRAN to do simulations of more complicated systems. Support packages were written to help out with routine chores like list processing, keeping track of simulated events, and statistical bookkeeping.

This approach was highly customizable and flexible (in terms of the kinds of models and manipulations possible), but also painfully tedious and error prone since models had to be coded pretty much from scratch every time. (Plus, if you dropped your deck of cards, it could take quite a while to reconstruct your "model.") For a more detailed history of discrete-event simulation languages, see Nance (1996) and Nance and Sargent (2003).

As a kind of descendant of simulating with general-purpose programming languages, people sometimes use spreadsheet software for some kinds of simulation. This has proven popular for static models, perhaps with add-ins to facilitate common operations and to provide higher-quality tools (like random-number generators) than what comes standard with spreadsheets. But for all but the very simplest dynamic models, the inherent limitations of spreadsheets make it at best awkward, and usually practically impossible, to use them for simulations of large, realistic, dynamic models. Section 2.7 briefly illustrates two examples of simulating with spreadsheets.

1.3.3 Simulation Languages

Special-purpose *simulation languages* like GPSS, Simscript, SLAM, and SIMAN appeared on the scene some time later and provided a much better framework for the kinds of simulations many people do. Simulation languages became very popular and are still in use.

Nonetheless, you still have to invest quite a bit of time to learn about their features and how to use them effectively. And, depending on the user interface provided, there can be picky, apparently arbitrary, and certainly frustrating syntactical idiosyncrasies that bedevil even old hands.

1.3.4 High-Level Simulators

Thus, several high-level "simulator" products emerged that are indeed very easy to use. They typically operate by intuitive graphical user interfaces, menus, and dialogs. You select from available simulation-modeling constructs, connect them, and run the model along with a dynamic graphical animation of system components as they move around and change.

However, the domains of many simulators are also rather restricted (like manufacturing or communications) and are generally not as flexible as you might like in order to build valid models of your systems. Some people feel that these packages may have gone too far up the software-hierarchy food chain and have traded away too much flexibility to achieve the ease-of-use goal.

1.3.5 Where Arena Fits In

Arena combines the ease of use found in high-level simulators with the flexibility of simulation languages and even all the way down to general-purpose procedural languages like the Microsoft® Visual Basic® programming system or C if you really want. It does this by providing alternative and interchangeable *templates* of graphical

simulation modeling and analysis *modules* that you can combine to build a fairly wide variety of simulation models. For ease of display and organization, modules are typically grouped into *panels* to compose a template. By switching panels, you gain access to a whole different set of simulation modeling constructs and capabilities. In most cases, modules from different panels can be mixed together in the same model.

Arena maintains its modeling flexibility by being fully *hierarchical*, as depicted in Figure 1-2. At any time, you can pull in low-level modules from the Blocks and Elements panel and gain access to simulation-language flexibility if you need to and mix in SIMAN constructs together with the higher-level modules from other templates (Arena is based on, and actually includes, the SIMAN simulation language; see Pedgen, Shannon, and Sadowski 1995 for a complete discussion of SIMAN). For specialized needs, like complex decision algorithms or accessing data from an external application, you can write pieces of your model in a procedural language like Visual Basic or C/C++. All of this, regardless of how high or low you want to go in the hierarchy, takes place in the same consistent graphical user interface.

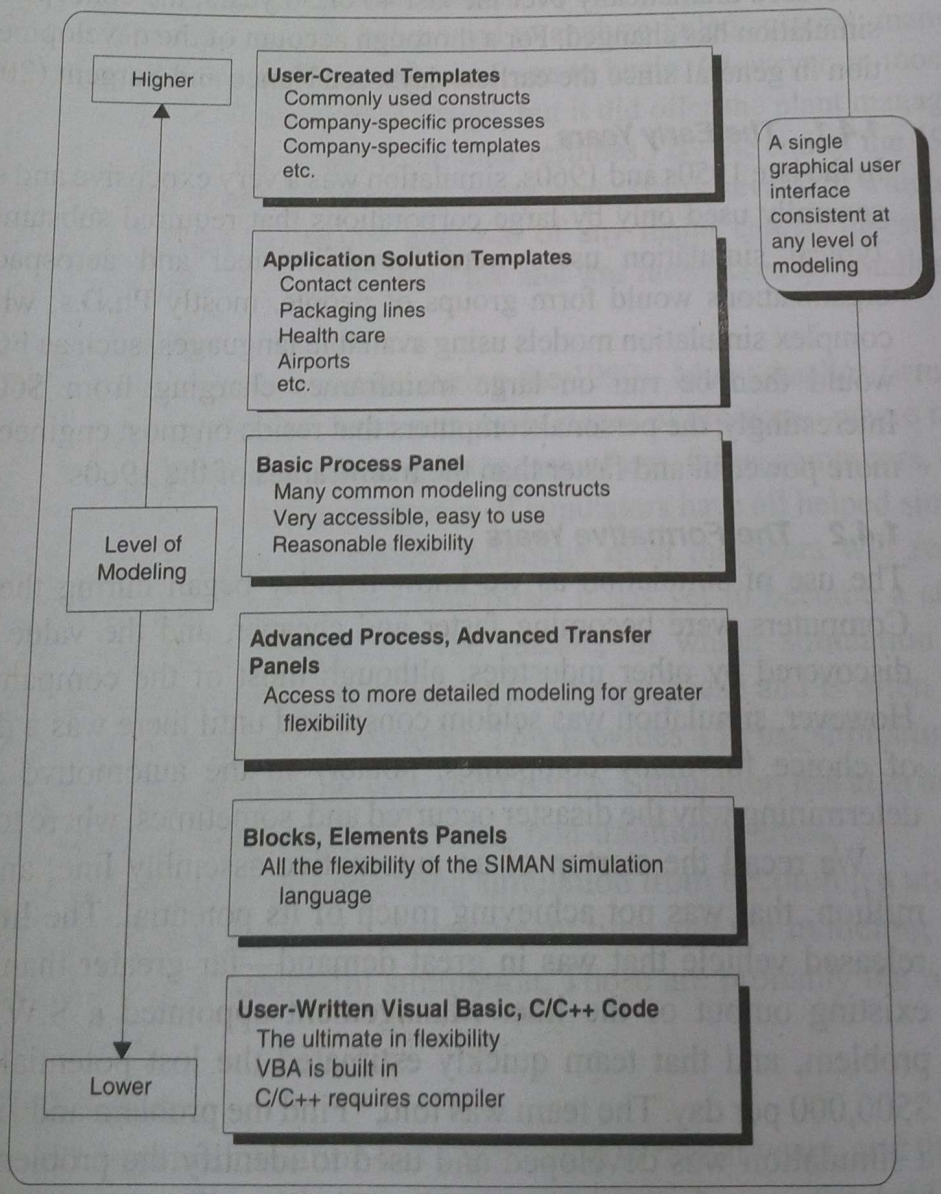


Figure 1-2. Arena's Hierarchical Structure

In fact, the modules in Arena are composed of SIMAN components; you can create your own modules and collect them into your own templates for various classes of systems. For instance, Rockwell Automation (formerly Systems Modeling) has built templates for general modeling (the Arena template, which is the primary focus of this book), high-speed packaging, contact centers, and other industries. Other people have built templates for their company in industries as diverse as mining, auto manufacturing, fast-food, and forest-resource management. In this way, you don't have to compromise between modeling flexibility and ease of use. While this textbook focuses on modeling with the Arena template, you can get a taste of creating your own modules in Chapter 10.

Further, Arena includes dynamic animation in the same work environment. It also provides integrated support, including graphics, for some of the statistical design and analysis issues that are part and parcel of a good simulation study.

1.4 When Simulations Are Used

Just as the capabilities and sophistication of simulation languages and packages have increased dramatically over the last 40 or 50 years, the concept of how and when to use simulation has changed. For a thorough account of the development of computer simulation in general since the earliest days, see Nance and Sargent (2003).

1.4.1 The Early Years

In the late 1950s and 1960s, simulation was a very expensive and specialized tool that was generally used only by large corporations that required substantial capital investments. Typical simulation users were found in steel and aerospace corporations. These organizations would form groups of people, mostly Ph.D.s, who would develop large, complex simulation models using available languages, such as FORTRAN. These models would then be run on large mainframes charging from $600 to $1,000 per hour. Interestingly, the personal computers that reside on most engineers' desks today are much more powerful and faster than the mainframes of the 1960s.

1.4.2 The Formative Years

The use of simulation as we know it today began during the 1970s and early 1980s. Computers were becoming faster and cheaper, and the value of simulation was being discovered by other industries, although most of the companies were still quite large. However, simulation was seldom considered until there was a disaster. It became the tool of choice for many companies, notably in the automotive and heavy industries, for determining why the disaster occurred and, sometimes, where to point the finger of blame.

We recall the startup of an automotive assembly line, an investment of over $100 million, that was not achieving much of its potential. The line was producing a newly released vehicle that was in great demand—far greater than could be satisfied by the existing output of the line. Management appointed a S.W.A.T. team to analyze the problem, and that team quickly estimated the lost potential profit to be in excess of $500,000 per day. The team was told, "Find the problem and fix it." In about three weeks, a simulation was developed and used to identify the problem, which turned out not to have been on the initial suspect list. The line was ultimately modified and did produce

according to specifications; unfortunately, by that time the competition was producing similar vehicles, and the additional output was no longer needed. Ironically, a simulation model had been used during the design of the assembly line to determine its feasibility. Unfortunately, many of the processes were new, and engineering had relied on equipment vendors to provide estimates of failures and throughputs. As is often the case, the vendors were extremely optimistic in their estimates. If the original design team had used the simulation to perform a good sensitivity analysis on these questionable data, the problem might have been uncovered and resolved well before implementation.

During this time, simulation also found a home in academia as a standard part of industrial engineering and operations research curricula. Its growing use in industry compelled universities to teach it more widely. At the same time, simulation began to reach into quantitative business programs, broadening the number and type of students and researchers exposed to its potential.

1.4.3 The Recent Past

During the late 1980s, simulation began to establish its real roots in business. A large part of this was due to the introduction of the personal computer and animation. Although simulation was still being used to analyze failed systems, many people were requesting simulations before production was to begin. (However, in most cases, it was really too late to affect the system design, but it did offer the plant manager and system designer the opportunity to spruce up their resumes.) By the end of the 1980s, the value of simulation was being recognized by many larger firms, several of which actually made simulation a requirement before approval of any major capital investment. However, simulation was still not in widespread use and was rarely used by smaller firms.

1.4.4 The Present

Simulation really began to mature during the 1990s. Many smaller firms embraced the tool, and it began to see use at the very early stages of projects—where it could have the greatest impact. Better animation, greater ease of use, faster computers, easy integration with other packages, and the emergence of simulators have all helped simulation become a standard tool in many companies. Although most managers will readily admit that simulation can add value to their enterprise, it has yet to become a standard tool that resides on everyone's computers. The manner in which simulation is used is also changing; it is being employed earlier in the design phase and is often being updated as changes are made to operating systems. This provides a living simulation model that can be used for systems analysis on very short notice. Simulation has also invaded the service industry where it is being applied in many non-traditional areas.

The major impediments preventing simulation from becoming a universally accepted and well-utilized tool are model-development time and the modeling skills required for the development of a successful simulation. Those are probably the reasons why you're reading this book!

1.4.5 The Future

The rate of change in simulation has accelerated in recent years, and there is every reason to believe that it will continue its rapid growth and cross the bridges to mainstream

acceptance. Simulation software has taken advantage of advancements in technology, enabling quicker model runs, higher-fidelity animations, and comprehensive data analysis. This trend must continue if simulation is to become a state-of-the-art and yet easy-to-use tool, required for effective decision-making. This evolution of simulation software has allowed for greater integration of simulation with other packages (like spreadsheets, databases, and word processors). It is now becoming possible to foresee the complete integration of simulation with other software packages that collect, store, and analyze system data at the front end along with software that helps control the system at the back end.

The Internet and intranets have had a tremendous impact on the way organizations conduct their business. Information sharing in real time is not only possible, but is becoming mandatory. Simulation tools are being developed to support distributed model building, distributed processing, and remote analysis of results. Soon, a set of enterprise-wide tools will be available to provide the ability for employees throughout an organization to obtain answers to critical and even routine questions.

In order to make simulation easier to use by more people, we will see more vertical products aimed at very specific markets. This will allow analysts to construct simulations easily, using modeling constructs designed for their industry or company with terminology that directly relates to their environment. These may be very specialized tools designed for very specific environments, but they should still have the capability to model any system activities that are unique to each simulation project. Some of these types of products are on the market today in application areas such as high-speed packaging, contact centers, communications, and business-process re-engineering.

Today's simulation projects concentrate on strategic decision making, typically around the design or redesign of complex systems. The next logical step is to use and evolve the same simulation to assist in operational decision making. This approach requires that the simulation model be kept current, but it also allows for examining and testing systems as market conditions, demands, and system processes change. As we progress to this next logical step, simulations will no longer be disposable or used only once, but will become a critical part of the operation of the ongoing system.

With the rapid advances being made in computers and software, it is very difficult to predict much about the simulations of the distant future, but even now we are seeing the development and implementation of features such as automatic statistical analysis, software that recommends system changes, simulations totally integrated with system operating software, and yes, even virtual reality.

CHAPTER 2

Fundamental Simulation Concepts

In this chapter, we introduce some of the underlying ideas, methods, and issues in simulation before getting into the Arena software itself in Chapter 3 and beyond. These concepts are the same across any kind of simulation software, and some familiarity with them is essential to understanding how Arena simulates a model you've built.

We do this mostly by carrying through a simple example, which is described in Section 2.1. In Section 2.2, we explore some options for dealing with the example model. Section 2.3 describes the various pieces of a simulation model, and Section 2.4 carries out the simulation (by hand), describing the fundamental organization and action. After that, two different simulation-modeling approaches are contrasted in Section 2.5, and the issue of randomness in both input and output is introduced in Section 2.6. Section 2.7 considers using spreadsheets to simulate, with both a static and a (very) simple dynamic model. Finally, Section 2.8 steps back and looks at what's involved in a simulation project, although this is taken up more thoroughly as part of Chapter 13.

By the end of this chapter, you'll understand the fundamental logic, structure, components, and management of a simulation modeling project. All of this underlies Arena and the richer models you'll build with it after reading subsequent chapters.

2.1 An Example

In this section, we describe the example system and decide what we'd like to know about its behavior and performance.

2.1.1 The System

Since a lot of simulation models involve waiting lines or *queues* as building blocks, we'll start with a very simple case of such a model representing a portion of a manufacturing facility. "Blank" parts arrive to a drilling center, are processed by a single drill press, and then leave; see Figure 2-1. If a part arrives and finds the drill press idle, its processing at the drill press starts right away; otherwise, it waits in a First-In First-Out (FIFO) queue. This is the *logical* structure of the model.

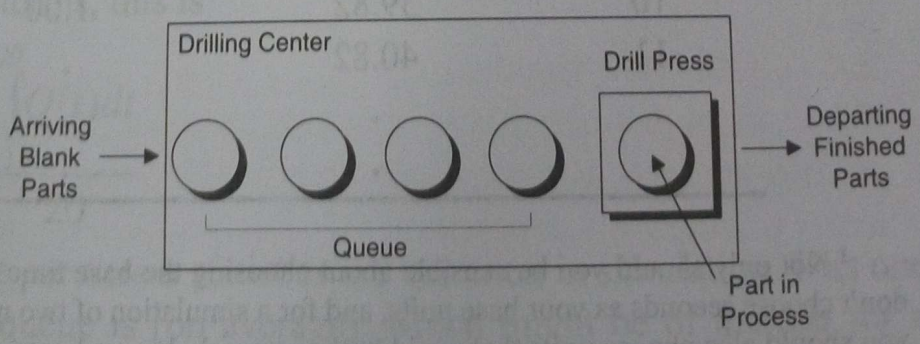


Figure 2-1. A Simple Processing System

You have to specify the *numerical* aspects as well, including how the simulation starts and stops. First, decide on the underlying "base" units with which time will be measured; we'll use minutes for all time measurements here. It doesn't logically matter what the time units are, so pick whatever is most appropriate, familiar, and convenient for your application.[1] You can express input time quantities in different units if it's customary or convenient, like minutes for mean service times but hours for mean machine-up times. However, for arithmetic and calculations, all times must be converted to the base time units if they're not already in it. Arena allows you to express input times in different units, but you must also declare the base time units into which all times are converted during the simulation run, and in which all time-based outputs are reported.

The system starts at time 0 minutes with no parts present and the drill press idle. This *empty-and-idle* assumption would be realistic if the system starts afresh each morning, but might not be so great as a model of the initial situation to simulate an ongoing operation.

The time durations that will make the simulation move are in Table 2-1. The (sequential) part number is in the first column, the second column has the time of arrival of each part, the third column gives the time *between* a part's arrival and that of the next part (called an *interarrival time*), and the service time (required to process on the drill press, not counting any time spent waiting in the queue) is in the last column. All times are in minutes. You're probably wondering where all these numbers came from; don't worry about that right now, and just pretend we observed them in the drilling center or that we brashly made them up.

Table 2-1. Arrival, Interarrival, and Service Times of Parts (in Minutes)

Part Number	Arrival Time	Interarrival Time	Service Time
1	0.00	1.73	2.90
2	1.73	1.35	1.76
3	3.08	0.71	3.39
4	3.79	0.62	4.52
5	4.41	14.28	4.46
6	18.69	0.70	4.36
7	19.39	15.52	2.07
8	34.91	3.15	3.36
9	38.06	1.76	2.37
10	39.82	1.00	5.38
11	40.82	.	.
.	.	.	.
.	.	.	.

[1] Not only should you be sensible about choosing the base time units (e.g., for a simulation of 20 years, don't choose seconds as your base units, and for a simulation of two minutes, don't measure time in days), but you should also choose units that avoid both extremely big and extremely tiny time values in the same model since, even with Arena's double-precision arithmetic, the computer might have trouble with round-off error.

We've decided that the simulation will stop at exactly time 20 minutes. If there are any parts present at that time (in service at the drill press or waiting in the queue), they are never finished.

2.1.2 Goals of the Study

Given a logical/numerical model like this, you next have to decide what output performance measures you want to collect. Here's what we decided to compute:

- The *total production* (number of parts that complete their service at the drill press and leave) during the 20 minutes of operation. Presumably, more is better.
- The *average waiting time in queue* of parts that enter service at the drill press during the simulation. This time in queue records only the time a part is waiting in the queue and not the time it spends being served at the drill press. If WQ_i is the waiting time in queue of the i^{th} part and it turns out that N parts leave the queue during the 20-minute run, this average is

$$\frac{\sum_{i=1}^{N} WQ_i}{N}.$$

(Note that since Part 1 arrives at time 0 to find the drill press idle, $WQ_1 = 0$ and $N \geq 1$ for sure so we don't have to worry about dividing by zero.) This is generally called a *discrete-time* (or *discrete-parameter*) statistic since it refers to data, in this case the waiting times $WQ_1, WQ_2, \ldots$, for which there is a natural first, second, . . . observation; in Arena, these are called *tally* statistics since values of them are "tallied" when they are observed (using a feature of the underlying SIMAN simulation language called *Tally*). From a performance standpoint, small is good.

- The *maximum waiting time in queue* of parts that enter service at the drill press during the simulation. This is a worst-case measure, which might be of interest in giving service-level guarantees to customers. Small is good.
- The *time-average number of parts waiting in the queue* (again, not counting any part in service at the drill press). By "time average," we mean a weighted average of the possible queue lengths (0, 1, 2, . . .) weighted by the proportion of time during the run that the queue was at that length. Letting $Q(t)$ be the number of parts in the queue at any time instant t, this time-average queue length is the total area under the $Q(t)$ curve, divided by the length of the run, 20. In integral-calculus terms, this is

$$\frac{\int_0^{20} Q(t)\,dt}{20}.$$

Such *time-persistent* statistics are common in simulation. This one indicates how long the queue is (on average), which might be of interest for allocating floor space.

- The *maximum number of parts that were ever waiting in the queue*. Actually, this might be a better indication of how much floor space is needed than is the time average if you want to be reasonably sure to have room at all times. This is another worst-case measure, and smaller is presumably better.
- The *average* and *maximum total time in system* of parts that finish being processed on the drill press and leave. Also called *cycle time*, this is the time that elapses between a part's arrival and its departure, so it's the sum of the part's waiting time in queue and its service time at the drill press. This is a kind of turnaround time, so smaller is better.
- The *utilization* of the drill press, defined as the proportion of time it is busy during the simulation. Think of this as another time-persistent statistic, but of the "busy" function

$$B(t)=\begin{cases}1 \text{ if the drill press is busy at time } t\\ 0 \text{ if the drill press is idle at time } t\end{cases}.$$

The utilization is the area under $B(t)$, divided by the length of the run:

$$\frac{\int_0^{20} B(t)dt}{20}.$$

Resource utilizations are of obvious interest in many simulations, but it's hard to say whether you "want" them to be high (close to 1) or low (close to 0). High is good since it indicates little excess capacity, but can also be bad since it might mean a lot of congestion in the form of long queues and slow throughput.

There are usually a lot of possible output performance measures, and it's probably a good idea to observe a lot of things in a simulation since you can always ignore things you have but can never look at things you don't have, plus sometimes you might find a surprise. The only downside is that collecting extraneous data can slow down execution of the simulation.

2.2 Analysis Options

With the model, its inputs, and its outputs defined, you next have to figure out how to get the outputs by transforming the inputs according to the model's logic. In this section, we'll briefly explore a few options for doing this.

2.2.1 Educated Guessing

While we're not big fans of guessing, a crude "back-of-the-envelope" calculation can sometimes lend at least qualitative insight (and sometimes not). How this goes, of course, completely depends on the situation (and on how good you are at guessing).

A possible first cut in our example is to look at the average inflow and processing rates. From Table 2-1, it turns out that the average of the ten interarrival times is 4.08 minutes, and the average of the ten service requirements is 3.46 minutes. This looks pretty hopeful, since parts are being served faster than they're arriving, at least on

average, meaning that the system has a chance of operating in a stable way over the long term and not "exploding." If these averages were *exactly* what happened for each part—no variation either way—then there would never be a queue and all waiting times in queue would be zero, a happy thought indeed. No matter how happy this thought might be, though, it's probably wrong since the data clearly show that there *is* variation in the interarrival and service times, which could create a queue sometimes; for example, if there happened to be some small interarrival times during a period when there also happened to be some large service times.

Suppose, on the other hand, that the averages in the input data in Table 2-1 had come out the other way, with the average interarrival time being smaller than the average service time. If this situation persisted, parts would, on average, be arriving faster than they could be served, implying heavy congestion (at least after a while, probably longer than the 20-minute run we have planned). Indeed, the system will explode over the long run— *not* a happy thought.

The truth, as usual, will probably be between the extremes. Clearly, guessing has its limits.

2.2.2 Queueing Theory

Since this is a queue, why not use queueing theory? It's been around for almost a century, and a lot of very bright people have worked very hard to develop it. In some situations, it can result in simple formulas from which you can get a lot of insight.

Probably the simplest and most popular object of queueing theory is the *M/M/1 queue*. The first "M" states that the arrival process is *Markovian*; that is, the interarrival times are independent and identically distributed "draws" from an exponential probability distribution (see Appendices B and C for a brief refresher on probability and distributions). The second "M" stands for the service-time distribution, and here it's also exponential. The "1" indicates that there's just a single server. So at least on the surface this looks pretty good for our model.

Better yet, most of our output performance measures can be expressed as simple formulas. For instance, the average waiting time in queue (expected from a long run) is

$$\frac{\mu_S^2}{\mu_A - \mu_S}$$

where μ_A is the expected value of the interarrival-time distribution and μ_S is the expected value of the service-time distribution (assuming that $\mu_A > \mu_S$ so the queue doesn't explode). So one immediate idea is to use the data to estimate μ_A and μ_S, then plug these estimates into the formula; for our data, we get $3.46^2/(4.08 - 3.46) = 19.31$ minutes.

Such an approach can sometimes give a reasonable order-of-magnitude approximation that might facilitate crude comparisons. But there are problems too (see Exercise 2-6):

- The estimates of μ_A and μ_S aren't exact, so there will be error in the result as well.
- The assumptions of exponential interarrival-time and service-time distributions are essential to deriving the formula above, and we probably don't satisfy these

assumptions. This calls into question the validity of the formula. While there are more sophisticated versions for more general queueing models, there will always be assumptions to worry about.

- The formula is for long-run performance, not the 20-minute period we want. This is typical of most (but not all) queueing theory.
- The formula doesn't provide any information on the natural variability in the system. This is not only a difficulty in analysis but might also be of inherent interest itself, as in the variability of production. (It's sometimes possible, though, to find other formulas that measure variability.)

Many people feel that queueing theory can prove valuable as a first-cut approximation to get an idea of where things stand and to provide guidance about what kinds of simulations might be appropriate at the next step in the project. We agree, but urge you to keep in mind the problems listed above and temper your interpretations accordingly.

2.2.3 Mechanistic Simulation

So all of this brings us back to simulation. By "mechanistic" we mean that the individual operations (arrivals, service by the drill press, etc.) will occur as they would in reality. The movements and changes of things in the simulation model occur at the right "time," in the right order, and have the right effects on each other and the statistical-accumulator variables.

In this way, simulation provides a completely concrete, "brute-force" way of dealing directly with the model. There's nothing mysterious about how it works—just a few basic ideas and then a whole lot of details and bookkeeping that software like Arena handles for you.

2.3 Pieces of a Simulation Model

We'll talk about the various parts of a simulation model in this section, all in reference to our example.

2.3.1 Entities

Most simulations involve "players" called *entities* that move around, change status, affect and are affected by other entities and the state of the system, and affect the output performance measures. Entities are the *dynamic* objects in the simulation—they usually are created, move around for a while, and then are disposed of as they leave. It's possible, though, to have entities that never leave but just keep circulating in the system. However, all entities have to be created, either by you or automatically by the software.

The entities for our example are the parts to be processed. They're created when they arrive, move through the queue (if necessary), are served by the drill press, and are then disposed of as they leave. Even though there's only one kind of entity in our example, there can be many independent "copies," or *realizations* of it in existence at a time, just as there can be many different individual parts of this type in the real system at a time.

Most entities represent "real" things in a simulation. You can have lots of different kinds of entities and many realizations of each kind of entity existing in the model at a time. For instance, you could have several different *kinds* of parts, perhaps requiring

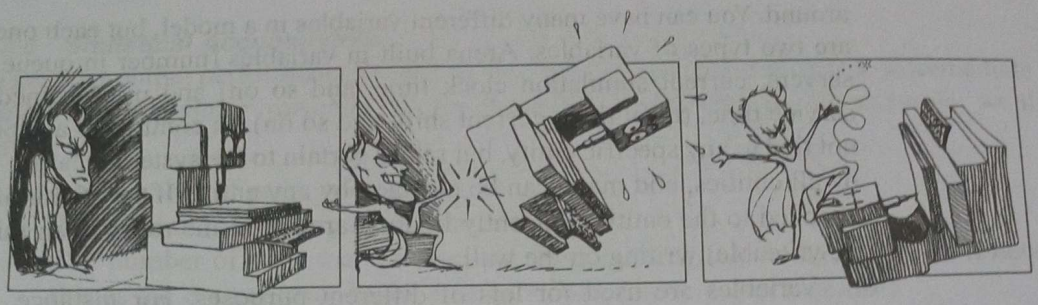


Figure 2-2. A Victorious Breakdown Demon

different processing and routing and having different priority; moreover, there could be several realizations of each kind of part floating around in the model at a time.

There are situations, though, where "fake" (or "logic") entities not corresponding to anything tangible can be conjured up to take care of certain modeling operations. For instance, one way to model machine failures is to create a "breakdown demon" (see Figure 2-2) that lurks in the shadows during the machine's up time, runs out and kicks the machine when it's supposed to break down, stands triumphantly over it until it gets repaired, then scurries back to the shadows and begins another lurking period representing the machine's next up time. A similar example is a "break angel" that arrives periodically and takes a server off duty. (For both of these examples, however, Arena has easier built-in modeling constructs that don't require creation of such fake entities.)

Figuring out what the entities are is probably the first thing you need to do in modeling a system.

2.3.2 Attributes

To individualize entities, you attach *attributes* to them. An attribute is a common characteristic of all entities, but with a specific value that can differ from one entity to another. For instance, our part entities could have attributes called `Due Date`, `Priority`, and `Color` to indicate these characteristics for each individual entity. It's up to you to figure out what attributes your entities need, name them, assign values to them, change them as appropriate, and then use them when it's time (that's all part of modeling).

The most important thing to remember about attributes is that their values are tied to specific entities. The same attribute will generally have different values for different entities, just as different parts have different due dates, priorities, and color codes. Think of an attribute as a tag attached to each entity, but what's written on this tag can differ across entities to characterize them individually. An analogy to traditional computer programming is that attributes are *local* variables—in this case, local to each individual entity.

Arena keeps track of some attributes automatically, but you may need to define, assign values to, change, and use attributes of your own.

2.3.3 (Global) Variables

A *variable* (or a *global* variable) is a piece of information that reflects some characteristic of your system, regardless of how many or what kinds of entities might be

around. You can have many different variables in a model, but each one is unique. There are two types of variables: Arena built-in variables (number in queue, number of busy servers, current simulation clock time, and so on) and user-defined variables (mean service time, travel time, current shift, and so on). In contrast to attributes, variables are not tied to any specific entity, but rather pertain to the system at large. They're accessible by all entities, and many can be changed by any entity. If you think of attributes as tags attached to the entities currently floating around in the room, then think of variables as (rewriteable) writing on the wall.

Variables are used for lots of different purposes. For instance, the time to move between any two stations in a model might be the same throughout the model, and a variable called `Transfer Time` could be defined and set to the appropriate value and then used wherever this constant is needed; in a modified model where this time is set to a different constant, you'd only need to change the definition of `Transfer Time` to change its value throughout the model. Variables can also represent something that changes *during* the simulation, like the number of parts in a certain subassembly area of the larger model, which is incremented by a part entity when it enters the area and decremented by a part when it leaves the area. Arena Variables can be vectors or matrices if it is convenient to organize the information as lists or two-dimensional tables of individual values.

Some built-in Arena variables for our model include the status of the drill press (busy or idle), the time (simulation clock), and the current length of the queue.

2.3.4 Resources

Entities often compete with each other for service from *resources* that represent things like personnel, equipment, or space in a storage area of limited size. An entity *seizes* (units of) a resource when available and *releases* it (or them) when finished. It's better to think of the resource as being given to the entity rather than the entity being assigned to the resource since an entity (like a part) could need simultaneous service from multiple resources (such as a machine and a person).

A resource can represent a group of several individual servers, each of which is called a *unit* of that resource. This is useful to model, for instance, several identical "parallel" agents at an airline ticketing counter. The number of available units of a resource can be changed during the simulation run to represent agents going on break or opening up their stations if things get busy. If a resource has multiple units, or a variable number of units, we have to generalize our definition in Section 2.1.2 of resource utilization to be the time-average number of units of the resource that are busy, divided by the time-average number of units of the resource that are available. In our example, there is just a single drill press, so this resource has a single unit available at all times.

2.3.5 Queues

When an entity can't move on, perhaps because it needs to seize a unit of a resource that's tied up by another entity, it needs a place to wait, which is the purpose of a *queue*. In Arena, queues have names and can also have capacities to represent, for instance, limited floor space for a buffer. You'd have to decide as part of your modeling how to handle an entity arriving at a queue that's already full.

2.3.6 Statistical Accumulators

To get your output performance measures, you have to keep track of various intermediate *statistical-accumulator variables* as the simulation progresses. In our example, we'll watch:

- The number of parts produced so far
- The total of the waiting times in queue so far
- The number of parts that have passed through the queue so far (since we'll need this as the denominator in the average waiting-time output measure)
- The longest time spent in queue we've seen so far
- The total of the time spent in the system by all parts that have departed so far
- The longest time in system we've seen so far
- The area so far under the queue-length curve $Q(t)$
- The highest level that $Q(t)$ has so far attained
- The area so far under the server-busy function $B(t)$

All of these accumulators should be initialized to 0. When something happens in the simulation, you have to update the affected accumulators in the appropriate way.

Arena takes care of most (but sometimes not all) of the statistical accumulation you're likely to want, so most of this will be invisible to you except for asking for it in some situations. But in our hand simulation, we'll do it all manually so you can see how it goes and will understand how to do it yourself in Arena if you need to.

2.3.7 Events

Now let's turn to how things work when we run our model. Basically, everything's centered around events. An *event* is something that happens at an instant of (simulated) time that might change attributes, variables, or statistical accumulators. In our example, there are three kinds of events:

- **Arrival:** A new part enters the system.
- **Departure:** A part finishes its service at the drill press and leaves the system.
- **The End:** The simulation is stopped at time 20 minutes. (It might seem rather artificial to anoint this as an event, but it certainly changes things, and this is one way to stop a simulation.)

In addition to the above events, there of course must be an initialization to set things up. We'll explain the logic of each event in more detail later.

Other things happen in our example model, but needn't be separate events. For instance, parts leave the queue and begin service at the drill press, which changes the system, but this only happens because of some other entity's departure, which is already an event.

To execute, a simulation has to keep track of the events that are supposed to happen in the (simulated) future. In Arena, this information is stored in an *event calendar*. We won't get into the details of the event calendar's data structure, but here's the idea: When the logic of the simulation calls for it, a *record* of information for a future event is placed on the event calendar. This event record contains identification of the entity involved, the event time, and the kind of event it will be. Arena places each newly scheduled event on

the calendar so that the next (soonest) event is always at the top of the calendar (that is, the new event record is *sorted* onto the calendar in increasing order of event times). When it's time to execute the next event, the top record is removed from the calendar and the information in this record is used to execute the appropriate logic; part of this logic might be to place one or more new event records onto the calendar. It's possible that, at a certain time, it doesn't make sense to have a certain event type scheduled (in our model, if the drill press is idle, you don't want a departure event to be scheduled), in which case there's just no record for that kind of event on the calendar, so it obviously can't happen next. Though this model doesn't require it, it's also possible to have several events of the same kind scheduled on the calendar at once, for different times and for different entities.

In a discrete-event model, the variables that describe the system don't change between successive events. Most of the work in event-driven simulation involves getting the logic right for what happens with each kind of event. As you'll see later, though, modeling with Arena usually gets you out of having to define this detailed event logic explicitly, although you can do so if you want in order to represent something very peculiar to your model that Arena isn't set up to handle directly.

2.3.8 Simulation Clock

The current value of time in the simulation is simply held in a variable called the *simulation clock*. Unlike real time, the simulation clock does not take on all values and flow continuously; rather, it lurches from the time of one event to the time of the next event scheduled to happen. Since nothing changes between events, there is no need to waste (real) time looking at (simulated) times that don't matter.

The simulation clock interacts closely with the event calendar. At initialization of the simulation, and then after executing each event, the event calendar's top record (always the one for the next event) is taken off the calendar. The simulation clock lurches forward to the time of that event (one of the data fields in the event record), and the information in the removed event record (entity identification, event time, and event type) is used to execute the event at that instant of simulated time. How the event is executed clearly depends on what kind of event it is as well as on the model state at that time, but in general could include updating variables and statistical accumulators, altering entity attributes, and placing new event records onto the calendar.

While we'll keep track of the simulation clock and event calendar ourselves in the hand simulation, these are clearly important pieces of any dynamic simulation, so Arena keeps track of them (the clock is a variable called TNOW in Arena).

2.3.9 Starting and Stopping

Important, but sometimes-overlooked, issues in a simulation are how it will start and stop. For our example, we've made specific assumptions about this, so it'll be easy to figure out how to translate them into values for attributes, variables, accumulators, the event calendar, and the clock.

Arena does a lot of things for you automatically, but it can't decide modeling issues like starting and stopping rules. You have to determine the appropriate starting conditions, how long a run should last, and whether it should stop at a particular time (as

we'll do at time 20 minutes) or whether it should stop when something specific happens (like as soon as 100 finished parts are produced). It's important to think about this and make assumptions consistent with what you're modeling; these decisions can have just as great an effect on your results as can more obvious things like values of input parameters (such as interarrival-time means, service-time variances, and the number of machines).

You should do *something* specific (and conscious) to stop the simulation with Arena, since it turns out that, in many situations, taking all the defaults will cause your simulation to run forever (or until you get sick of waiting and kill it, whichever comes first).

2.4 Event-Driven Hand Simulation

We'll let you have the gory details of the hand simulation in this section, after outlining the action and defining how to keep track of things.

2.4.1 Outline of the Action

Here's roughly how things go for each event:

- **Arrival:** A new part shows up.
  - Schedule the next new part to arrive later, at the next arrival time, by placing a new event record for it onto the event calendar.
  - Update the time-persistent statistics (between the last event and now).
  - Store the arriving part's time of arrival (the current value of the clock) in an attribute, which will be needed later to compute its total time in system and possibly the time it spends waiting in the queue.
  - If the drill press is idle, the arriving part goes right into service (experiencing a time in queue of zero), so the drill press is made busy and the end of this part's service is scheduled. Tally this part's time in queue (zero).
  - On the other hand, if the drill press is already busy with another part, the arriving part is put at the end of the queue and the queue-length variable is incremented.
- **Departure:** The part being served by the drill press is done and ready to leave.
  - Increment the number-produced statistical accumulator.
  - Compute and tally the total time in system of the departing part by taking the current value of the clock minus the part's arrival time (stored in an attribute during the Arrival event).
  - Update the time-persistent statistics.
  - If there are any parts in queue, take the first one out, compute and tally its time in queue (which is now ending), and begin its service at the drill press by scheduling its departure event (that is, place it on the event calendar).
  - On the other hand, if the queue is empty, then make the drill press idle. Note that in this case, there's no departure event scheduled on the event calendar.
- **The End:** The simulation is over.
  - Update the time-persistent statistics to the end of the simulation.
  - Compute and report the final summary output performance measures.

After each event (except the end event), the event calendar's top record is removed, indicating what event will happen next and at what time. The simulation clock is advanced to that time, and the appropriate logic is carried out.

2.4.2 Keeping Track of Things

All the calculations for the hand simulation are detailed in Table 2-2. Traces of $Q(t)$ and $B(t)$ over the whole simulation are in Figure 2-3. Each row in Table 2-2 represents an event concerning a particular part (in the first column) at time t (second column), and the situation *after* completion of the logic for that event (in the other columns). The other column groups are:

- **Event:** This describes what just happened; Arr and Dep refer respectively to an arrival and a departure.
- **Variables:** These are the values of the number $Q(t)$ of parts in queue and the server-busy function $B(t)$.
- **Attributes:** Each arriving entity's arrival time is assigned when it arrives and is carried along with it throughout. If a part is in service at the drill press, its arrival time is at the right edge of the column and is not enclosed in parentheses. The arrival times of any parts in the queue, in right-to-left order (to agree with Figure 2-1), extend back toward the left and are in parentheses. For instance, at the end of the run, the part in service arrived at time 18.69, and the part that's first (and only) in the queue arrived at time 19.39. We have to keep track of these to compute the time in queue of a part when it enters service at the drill press after having waited in the queue as well as the total time in system of a part when it leaves.
- **Statistical Accumulators:** We have to initialize and then update these as we go along to watch what happens. They are:

  P = the total number of parts produced so far
  N = the number of entities that have passed through the queue so far
  ΣWQ = the sum of the queue times that have been observed so far
  WQ^* = the maximum time in queue observed so far
  ΣTS = the sum of the total times in system that have been observed so far
  TS^* = the maximum total time in system observed so far
  $\int Q$ = the area under the $Q(t)$ curve so far
  Q^* = the maximum value of $Q(t)$ so far
  $\int B$ = the area under the $B(t)$ curve so far

- **Event Calendar:** These are the event records as described earlier. Note that, at each event time, the top record on the event calendar becomes the first three entries under "Just-Finished Event" at the left edge of the entire table in the next row, at the next event time.

Table 2-2. Record of the Hand Simulation

Just-Finished Event			Variables		Attributes		Statistical Accumulators								Event Calendar
Entity No.	Time t	Event Type	$Q(t)$	$B(t)$	Arrival Times: (In Queue)	In Service	P	N	ΣWQ	WQ^*	ΣTS	TS^*	$\int Q$	Q^*	$\int B$	[Entity No., Time, Type]
–	0.00	Init	0	0	( )	–	0	0	0.00	0.00	0.00	0.00	0.00	0	0.00	[1, 0.00, Arr] [–, 20.00, End]
1	0.00	Arr	0	1	( )	0.00	0	1	0.00	0.00	0.00	0.00	0.00	0	0.00	[2, 1.73, Arr] [1, 2.90, Dep] [–, 20.00, End]
2	1.73	Arr	1	1	(1.73)	0.00	0	1	0.00	0.00	0.00	0.00	0.00	1	1.73	[1, 2.90, Dep] [3, 3.08, Arr] [–, 20.00 End]
1	2.90	Dep	0	1	( )	1.73	1	2	1.17	1.17	2.90	2.90	1.17	1	2.90	[3, 3.08, Arr] [2, 4.66, Dep] [–, 20.00, End]
3	3.08	Arr	1	1	(3.08)	1.73	1	2	1.17	1.17	2.90	2.90	1.17	1	3.08	[4, 3.79, Arr] [2, 4.66, Dep] [–, 20.00, End]
4	3.79	Arr	2	1	(3.79, 3.08)	1.73	1	2	1.17	1.17	2.90	2.90	1.88	2	3.79	[5, 4.41, Arr] [2, 4.66, Dep] [–, 20.00, End]
5	4.41	Arr	3	1	(4.41, 3.79, 3.08)	1.73	1	2	1.17	1.17	2.90	2.90	3.12	3	4.41	[2, 4.66, Dep] [6, 18.69, Arr] [–, 20.00, End]
2	4.66	Dep	2	1	(4.41, 3.79)	3.08	2	3	2.75	1.58	5.83	2.93	3.87	3	4.66	[3, 8.05, Dep] [6, 18.69, Arr] [–, 20.00, End]
3	8.05	Dep	1	1	(4.41)	3.79	3	4	7.01	4.26	10.80	4.97	10.65	3	8.05	[4, 12.57, Dep] [6, 18.69, Arr] [–, 20.00, End]
4	12.57	Dep	0	1	( )	4.41	4	5	15.17	8.16	19.58	8.78	15.17	3	12.57	[5, 17.03, Dep] [6, 18.69, Arr] [–, 20.00, End]
5	17.03	Dep	0	0	( )	–	5	5	15.17	8.16	32.20	12.62	15.17	3	17.03	[6, 18.69, Arr] [–, 20.00 End]
6	18.69	Arr	0	1	( )	18.69	5	6	15.17	8.16	32.20	12.62	15.17	3	17.03	[7, 19.39, Arr] [–, 20.00, End] [6, 23.05, Dep]
7	19.39	Arr	1	1	(19.39)	18.69	5	6	15.17	8.16	32.20	12.62	15.17	3	17.73	[–, 20.00, End] [6, 23.05, Dep] [8, 34.91, Arr]
–	20.00	End	1	1	(19.39)	18.69	5	6	15.17	8.16	32.20	12.62	15.78	3	18.34	[6, 23.05, Dep] [8, 34.91, Arr]

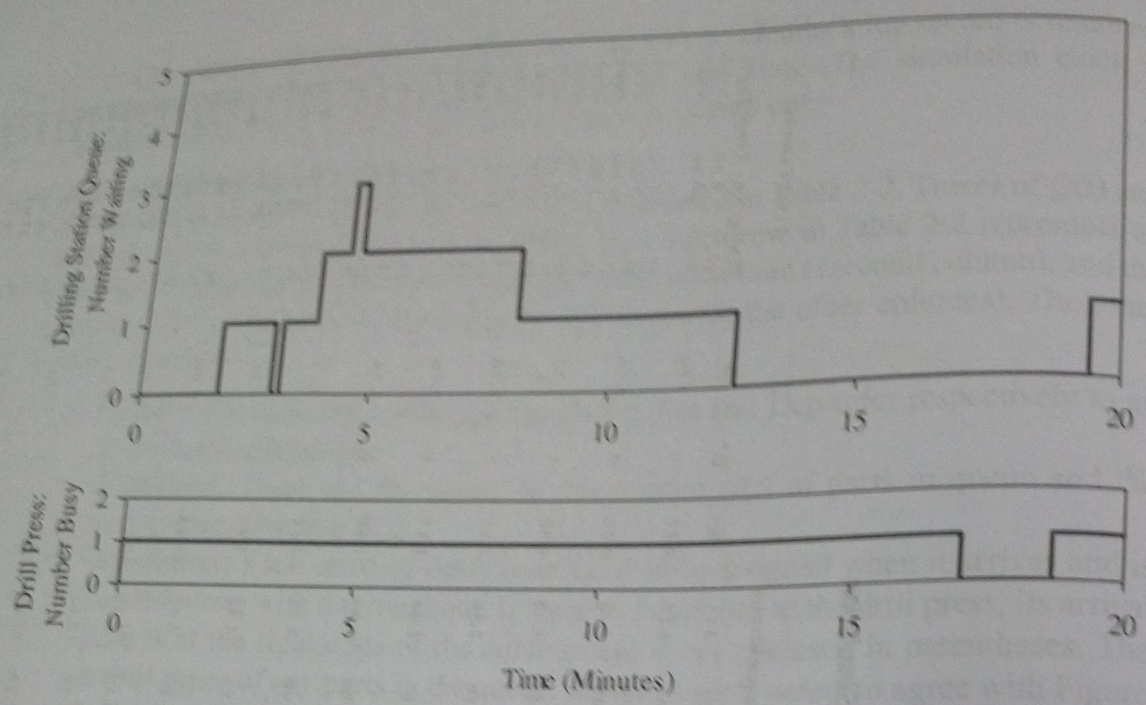


Figure 2-3. Time-Persistent Curves for the Hand Simulation

2.4.3 Carrying It Out

Here's a brief narrative of the action:

- **$t = 0.00$, Init:** The model is initialized, with all variables and accumulators set to 0, the queue empty, the drill press idle, and the event calendar primed with the first arrival happening at time 0.00 and the end of the simulation scheduled for time 20.00. To see what happens next, just take the first event record off the event calendar—the arrival of Entity 1 at time 0.00.
- **Entity 1, $t = 0.00$, Arr:** The next (second) part arrival is scheduled by creating a Part 2 entity, setting its Arrival Time to the current time (0) plus its interarrival time (1.73, from Table 2-1), and placing it on the event calendar. Meanwhile, the part arriving now (Part 1) makes the drill press busy, and the arrival time, 0.00, of this part is stored in its attribute (not in parentheses). The queue is still empty because this part finds the drill press idle and begins service immediately (so the parentheses to contain the times of arrival of parts in queue have nothing between them). Since the entity passed through the queue (with time in queue of 0), N is incremented, ΣWQ is augmented by the waiting time in queue (0), and we check to see if a new maximum waiting time in queue has occurred (no). There has been no production yet, so P stays at 0. No total times in system have yet been observed, so ΣTS and TS^* are unchanged. The time-persistent statistics $\int Q$, Q^*, and $\int B$ remain at 0 since no time has yet passed. Referring to Table 2-1, the service time for Part 1 is set to 2.90 minutes, and the Part 1 entity is returned to the event calendar. Taking the top record off the event calendar, the next event will be the arrival of Entity 2 at time 1.73.

- **Entity 2, $t = 1.73$, Arr:** The next (third) part arrival is scheduled by creating a Part 3 entity, setting its Arrival Time to the current time (1.73) plus its interarrival time (1.35, from Table 2-1), and placing it on the event calendar, which schedules the arrival of this part at time 1.73 + 1.35 = 3.08. Meanwhile, the part arriving now (Part 2) finds the drill press already busy (with Part 1), so it must queue up. The drill press is still busy, so $B(t)$ stays at 1, but the queue length $Q(t)$ is incremented from 0 to 1. Now there is a queue, and the arrival time (the time now, which is 1.73) of the part in the queue (the currently arriving Part 2) is stored as an attribute of that part in the queue (in parentheses); Part 1, which arrived at time 0.00, is still in service. This event has resulted in no new production and no new time-in-queue observations, so P, N, ΣWQ, WQ^*, ΣTS, and TS^* are unchanged. $\int Q$ is augmented by $0 \times (1.73 - 0.00) = 0$ and $\int B$ is augmented by $1 \times (1.73 - 0.00) = 1.73$. Since the new value of $Q(t)$ is 1, which is greater than the former $Q^* = 0$, we set $Q^* = 1$ as the new maximum queue length observed so far. We don't update the time of the next departure, since it's still (correctly) scheduled as the departure time of Part 1, which arrived earlier and is now partially processed. The next event will be the departure of Part 1 at time 2.90.
- **Entity 1, $t = 2.90$, Dep:** Since this is a departure event, we don't need to schedule the next arrival—it's already on its way, with the (correct) arrival event [3, 3.08, Arr] already on the event calendar. Part 1 is now done being processed by the drill press and ready to depart. Since there is a queue, the drill press will stay busy, so $B(t)$ remains at 1, but $Q(t)$ is decremented (from 1 to 0) and the queue becomes empty (as Part 2, which arrived at time 1.73, leaves it to enter service). Part 2's waiting time in queue of duration 2.90 – 1.73 = 1.17 has now been completed, which is added into ΣWQ, and N is incremented; this is a new maximum waiting time in queue, so WQ^* is redefined to 1.17. A finished part (Part 1) has been produced so P is incremented and the total time in system for Part 1 is computed as 2.90 – 0.00 = 2.90, which is added into ΣTS; this is a new maximum total time in system, so TS^* is updated to 2.90. We augment $\int Q$ by $1 \times (2.90 - 1.73) = 1.17$, and $\int B$ is augmented by the same amount. No new maximum for $Q(t)$ has been realized, so Q^* is unchanged at 1. From Table 2-1, the next service time is 1.76, so the next departure (of Part 2, which is now entering service) will be at time 2.90 + 1.76 = 4.66. The next event is the arrival of Part 3 at time 3.08.
- **Entity 3, $t = 3.08$, Arr:** The next (fourth) part arrival is scheduled by creating a Part 4 entity, setting its Arrival Time to 3.08 + 0.71 = 3.79 (0.71 is the next interarrival time in Table 2-1), and placing it on the event calendar. Meanwhile, the part arriving now (Part 3) finds the drill press already busy (with Part 2), so Part 3 queues up. The drill press is still busy, so $B(t)$ stays at 1, but the queue length $Q(t)$ is incremented from 0 to 1. The arrival time (3.08) of the part in queue (the currently arriving Part 3) is stored as an attribute of that part; Part 2, which arrived at time 1.73, is still in service. With this event, we're not experiencing any new production or time-in-queue observations, so P, N, ΣWQ, WQ^*, ΣTS, and TS^* are unchanged. $\int Q$ is augmented by $0 \times (3.08 - 2.90) = 0$ and $\int B$ is augmented by $1 \times (3.08 - 2.90) = 0.18$. The new value of $Q(t)$ is 1, which is no more than the

former $Q^* = 1$, so we leave $Q^* = 1$ unchanged. We don't update the time of the next departure, since it's still (correctly) scheduled as the departure time of Part 2, which arrived earlier and is now partially processed. The next event will be the arrival of Part 4 at time 3.79.

- **Entity 4, $t = 3.79$, Arr:** The next (fifth) part arrival is scheduled for time 3.79 + 0.62 = 4.41. The part arriving now (Part 4) queues up since the drill press is now busy; $B(t)$ stays at 1, and the queue length $Q(t)$ is incremented to 2. The arrival time (3.79) of Part 4, now arriving, is stored as an attribute of that part, and placed at the end of the queue (the queue is FIFO); Part 2, which arrived at time 1.73, is still in service. We're not experiencing any new production or time-in-queue observations, so P, N, ΣWQ, WQ^*, ΣTS, and TS^* are unchanged. $\int Q$ is augmented by $1 \times (3.79 - 3.08) = 0.71$ and $\int B$ is augmented by $1 \times (3.79 - 3.08) = 0.71$. The new value of $Q(t)$ is 2, which exceeds the former $Q^* = 1$, so we now set $Q^* = 2$. We don't update the time of the next departure, since it's still (correctly) scheduled as the departure time of Part 2. The next event will be the arrival of Part 5 at time 4.41.
- **Entity 5, $t = 4.41$, Arr:** The next (sixth) part arrival is scheduled for time 4.41 + 14.28 = 18.69. The part arriving now (Part 5) joins the end of the queue, $B(t)$ stays at 1, the queue length $Q(t)$ is incremented to 3, and Part 2 continues in service. As in the preceding event, P, N, ΣWQ, WQ^*, ΣTS, and TS^* are unchanged. $\int Q$ is augmented by $2 \times (4.41 - 3.79) = 1.24$ and $\int B$ is augmented by $1 \times (4.41 - 3.79) = 0.62$. $Q(t)$ has now risen to 3, a new maximum, so we set $Q^* = 3$. The time of the next departure is still (correctly) scheduled as the departure time of Part 2 at time 4.66, which now floats to the top of the event calendar and will be the next event to occur.
- **Entity 2, $t = 4.66$, Dep:** Part 2 now is done and is ready to depart. Since there is a queue, the drill press will stay busy so $B(t)$ remains at 1, but $Q(t)$ is decremented to 2. The queue advances, and the first part in it (Part 3, which arrived at time 3.08) will enter service. A waiting time in queue of duration $4.66 - 3.08 = 1.58$ for Part 3 has now been completed, which is added into ΣWQ, and N is incremented; this is a new maximum waiting time in queue, so WQ^* is changed to 1.58. A new part has been produced so P is incremented to 2 and the total time in system of Part 2 is computed as $4.66 - 1.73 = 2.93$, which is added into ΣTS; this is a new maximum total time in system, so TS^* is changed to 2.93. $\int Q$ is augmented by $3 \times (4.66 - 4.41) = 0.75$ and $\int B$ is augmented by $1 \times (4.66 - 4.41) = 0.25$. No new maximum for $Q(t)$ has been realized, so Q^* is unchanged. The next departure (of Part 3, which is now entering service) is scheduled for time 4.66 + 3.39 = 8.05; no update to the time of the next arrival is needed. The next event is the departure of Entity 3 at time 8.05.
- **Entity 3, $t = 8.05$, Dep:** Part 3 departs. Part 4, first in queue, enters service, adds its waiting time in queue $8.05 - 3.79 = 4.26$ (a new maximum) into ΣWQ, and N is incremented; $B(t)$ stays at 1 and $Q(t)$ is decremented to 1. Part 3's total time in system is $8.05 - 3.08 = 4.97$ (a new maximum), which is added into ΣTS, and P is incremented. $\int Q$ is augmented by $2 \times (8.05 - 4.66) = 6.78$ and $\int B$ is augmented by

$1 \times (8.05 - 4.66) = 3.39$; Q^* is unchanged. The next departure (of Part 4, which is now entering service) is scheduled for time $8.05 + 4.52 = 12.57$; no update to the time of the next arrival is needed. The next event is the departure of Entity 4 at time 12.57.

- **Entity 4, $t = 12.57$, Dep:** Part 4 departs. Part 5 enters service, adds its waiting time in queue $12.57 - 4.41 = 8.16$ (a new maximum) into ΣWQ, and N is incremented; $B(t)$ stays at 1 and $Q(t)$ is decremented to 0 (so the queue becomes empty, though a part is in service). Part 4's total time in system is $12.57 - 3.79 = 8.78$ (a new maximum), which is added into ΣTS, and P is incremented. $\int Q$ is augmented by $1 \times (12.57 - 8.05) = 4.52$ and $\int B$ is augmented by the same amount; Q^* is unchanged. The next departure (of Part 5, which is now entering service) is scheduled for time $12.57 + 4.46 = 17.03$; no update to the time of the next arrival is needed. The next event is the departure of Entity 5 at time 17.03.
- **Entity 5, $t = 17.03$, Dep:** Part 5 departs. But since the queue is empty, the drill press becomes idle ($B(t)$ is set to 0) and there is no new waiting time in queue observed, so ΣWQ, WQ^*, and N remain unchanged. Part 5's total time in system is $17.03 - 4.41 = 12.62$ (a new maximum), which is added into ΣTS, and P is incremented. $\int Q$ is augmented by $0 \times (17.03 - 12.57) = 0$ and $\int B$ is augmented by $1 \times (17.03 - 12.57) = 4.46$; Q^* is unchanged. Since there is not now a part in service at the drill press, the event calendar is left without a departure event scheduled; no update to the time of the next arrival is needed. The next event is the arrival of Entity 6 at time 18.69.
- **Entity 6, $t = 18.69$, Arr:** This event processes the arrival of Part 7 to the system, which is now empty of parts and the drill press is idle, so this is really the same scenario as in the arrival of Part 1 at time 0; indeed, the "experience" of Part 7 is (probabilistically) identical to that of Part 1. The next (seventh) part arrival is scheduled for time $18.69 + 0.70 = 19.39$. Meanwhile, the arriving Part 6 makes the drill press busy, but the queue is still empty because this part finds the drill press idle and begins service immediately. The waiting time in queue of Part 6 is 0, so N is incremented to 6, and ΣWQ is "augmented" by 0, which is certainly not a new maximum waiting time in queue. No part is departing, so there is no change in P, ΣTS, or TS^*. Since $Q(t)$ and $B(t)$ have both been at 0 since the previous event, there is no numerical change in $\int Q$, $\int B$, or Q^*. The departure of Part 6 is scheduled for time $18.69 + 4.36 = 23.05$ (so will not occur since this is after the simulation end time, as reflected in the order of the updated event calendar). The next event is the arrival of Part 7 at time 19.39.
- **Entity 7, $t = 19.39$, Arr:** The next (eighth) part arrival is scheduled for time $19.39 + 15.52 = 34.91$, which is beyond the termination time (20), so will not happen. Part 8 queues up, $B(t)$ stays at 1, and $Q(t)$ is incremented to 1 (not a new maximum). Since there is no new waiting time in queue being observed, N, ΣWQ, and WQ^* are unchanged; since no part is departing, P, ΣTS, and TS^* are unchanged. $\int Q$ is augmented by $0 \times (19.39 - 18.69) = 0$ and $\int B$ is augmented by $1 \times (19.39 - 18.69) = 0.70$. The next departure is already properly scheduled. The next event will be the end of the simulation at time 20.

Table 2-3. Final Output Performance Measures from the Hand Simulation

Performance Measure	Value
Total production	5 parts
Average waiting time in queue	2.53 minutes per part (6 parts)
Maximum waiting time in queue	8.16 minutes
Average total time in system	6.44 minutes per part (5 parts)
Maximum total time in system	12.62 minutes
Time-average number of parts in queue	0.79 part
Maximum number of parts in queue	3 parts
Drill-press utilization	0.92 (dimensionless proportion)

- **$t = 20.00$, The End:** The only task here is to update the areas $\int Q$ and $\int B$ to the end of the simulation, which in this state adds $1 \times (20.00 - 19.39) = 0.61$ to both of them.

The bottom row of Table 2-2 shows the ending situation, including the final values of the statistical accumulators.

2.4.4 Finishing Up

The only cleanup required is to compute the final values of the output performance measures:

- The average waiting time in queue is $\Sigma WQ/N = 15.17/6 = 2.53$ minutes per part.
- The average total time in system is $\Sigma TS/P = 32.20/5 = 6.44$ minutes per part.
- The time-average length of the queue is $\int Q/t = 15.78/20 = 0.79$ part (t here is the final value, 20.00, of the simulation clock).
- The utilization of the drill press is $\int B/t = 18.34/20 = 0.92$.

Table 2-3 summarizes all the final output measures together with their units of measurement.

During the 20 minutes, we produced five finished parts; the waiting time in queue, total times in system, and queue length do not seem too bad; and the drill press was busy 92% of the time. These values are considerably different from what we might have guessed or obtained via an oversimplified queueing model (see Exercise 2-6).

2.5 Event- and Process-Oriented Simulation

The hand simulation we struggled through in Section 2.4 uses the *event orientation* since the modeling and computational work is centered around the events, when they occur, and what happens when they do. This allows you to control everything; have complete flexibility with regard to attributes, variables, and logic flow; and to know the state of everything at any time. You easily see how this could be coded up in any programming language or maybe with macros in a spreadsheet (for very simple models only), and people have done this a lot. For one thing, computation is pretty quick with a custom-written, event-oriented code. While the event orientation seems simple enough in

principle (although not much fun by hand) and has some advantages, you can imagine that it becomes very complicated for large models with lots of different kinds of events, entities, and resources.

A more natural way to think about many dynamic simulations is to take the viewpoint of a "typical" entity as it works its way through the model, rather than the omniscient orientation of the master controller keeping track of all events, entities, attributes, variables, and statistical accumulators as we did in the event-oriented hand simulation. This alternative view centers on the *processes* that entities undergo, so is called the *process orientation*. As you'll see, this is strongly analogous to another common modeling tool—namely, flowcharting. In this view, we might model the hand-simulated example in steps like this (put yourself in the position of a typical part entity):

- Create yourself (a new entity arrives).
- Write down what time it is now on one of your attributes so you'll know your arrival time later for the waiting-time-in-queue and total-time-in-system computations.
- Put yourself at the end of the queue.
- Wait in the queue until the drill press becomes free (this wait could be of 0 duration, if you're lucky enough to find the drill press idle when you arrive).
- Seize the drill press (and take yourself out of the queue).
- Compute and tally your waiting time in queue.
- Stay put, or *delay* yourself, for an amount of time equal to your service requirement.
- Release the drill press (so other entities can seize it).
- Increment the production-counter accumulator on the wall and tally your total time in system.
- Dispose of yourself and go away.

This is the sort of "program" you write with a process-oriented simulation language like SIMAN and is also the view of things normally taken by Arena (which actually translates your flowchart-like description into a SIMAN model to be run, though you don't need to get into that unless you want to; see Section 7.1.6). It's a much more natural way to think about modeling, and (importantly) big models can be built without the extreme complexity they'd require in an event-oriented program. It does, though, require more behind-the-scenes support for chores like time advance, keeping track of time-persistent statistics (which didn't show up in the process-oriented logic), and output-report generation. Simulation software like Arena provides this support as well as a rich variety of powerful modeling constructs that enable you to build complicated models relatively quickly and reliably.

Most discrete-event simulations are actually *executed* in the event orientation, even though you may never see it if you do your modeling in the process orientation. Arena's hierarchical nature allows you to get down into the event orientation if you need to in order to regain the control to model something peculiar, and in that case you have to think (and code) with event-oriented logic as we did in the hand simulation.

Because of its ease and power, process-oriented logic has become very popular and is the approach we'll take from now on. However, it's good to have some understanding of what's going on under the hood, so we first made you suffer through the laborious event simulation.

2.6 Randomness in Simulation

In this section, we'll discuss how (and why) you model randomness in a simulation model's input and the effect this can have on your output. We'll need some probability and statistics here, so this might be a good time to take a quick glance at (or a painstaking review of) Appendix B to review some basic ideas, terminology, and notation.

2.6.1 Random Input, Random Output

The simulation in Section 2.4 used the input data in Table 2-1 to drive the simulation recorded in Table 2-2, resulting in the numerical output performance measures reported in Table 2-3. This might be what happened from 8:00 to 8:20 on some particular Monday morning, and if that's all you're interested in, you're done.

But you're probably interested in more, like what you'd expect to see on a "typical" morning and how the results might differ from day to day. And since the arrival and service times of parts on other days would probably differ from those in Table 2-1, the trace of the action in Table 2-2 will likely be changed, so the numerical output performance measures will probably be different from what we got in Table 2-3. Therefore, a single run of the example just won't do since we really have no idea how "typical" our results are or how much variability might be associated with them. In statistical terms, what you get from a single run of a simulation is a *sample of size one*, which just isn't worth much. It would be pretty unwise to put much faith in it, much less make important decisions based on it alone. If you rolled a die once and it came up a four, would you be willing to conclude that all the other faces were four as well?

So random input looks like a curse. But you must often allow for it to make your model a valid representation of reality, where there may also be considerable uncertainty. The way people usually model this, instead of using a table of numerical input values, is to specify *probability distributions* from which observations are *generated* (or *drawn* or *sampled*) and drive the simulation with them. We'll talk in Section 4.4 about how you can determine these input probability distributions using the Arena Input Analyzer. Arena internally handles generation of observations from distributions you specify. Not only does this make your model more realistic, but it also sets you free to do more simulation than you might have observed data for and to explore situations that you didn't actually observe. As for the tedium of generating the input observations and doing the simulation logic, that's exactly what Arena (and computers) like to do for you.

But random input induces randomness in the output too. We'll explore this a little bit in the remainder of this section, but will take it up more fully in Chapter 6, Section 7.2, and Chapter 12 and show you how to use the Arena Output Analyzer, Process Analyzer, and OptQuest®, from OptTek Systems, Inc., to help interpret and appropriately cope with randomness in the output.

2.6.2 Replicating the Example

It's time to confess: We generated the input values in Table 2-1 from probability distributions in Arena. The interarrival times came from an exponential distribution with a mean of 5 minutes, and the service times came from a triangular distribution with a minimum of 1 minute, mode of 3 minutes, and maximum of 6 minutes. (You'll see in Chapter 3 how all this works in Arena.) See Appendix C for a description of Arena's probability distributions.

So instead of just the single 20-minute run, we could make several (we'll do five) independent, statistically identical 20-minute runs and investigate how the results change from run to run, indicating how things in reality might change from morning to morning. Each run starts and stops according to the same rules and uses the same input-parameter settings (that's the "statistically identical" part), but uses separate input random numbers (that's the "independent" part) to generate the actual interarrival and service times used to drive the simulation. Another thing we must do is *not* carry over any in-queue or in-process parts from the end of one run to the beginning of the next as they would introduce a link, or correlation, between one run and the next; any such luckless parts just go away. Such independent and statistically identical runs are called *replications* of the simulation, and Arena makes it very easy for you to carry them out—just enter the number of replications you want into a dialog box on your screen. You can think of this as having five replications of Table 2-1 for the input values, each one generating a replication of the simulation record in Table 2-2, resulting in five replications of Table 2-3 for all the results.

We wish you'd admire (or pity) us for slugging all this out by hand, but we really just asked Arena to do it for us; the results are in Table 2-4. The column for Replication 1 is the same as what's in Table 2-3, but you can see that there can be substantial variation across replications, just as things vary across days in the factory.

Table 2-4. Final Output Performance Measures from Five Replications of the Hand Simulation

	Replication					Sample		95%
Performance Measure	**1**	**2**	**3**	**4**	**5**	**Avg.**	**Std. Dev.**	**Half Width**
Total production	5	3	6	2	3	3.80	1.64	2.04
Average waiting time in queue	2.53	1.19	1.03	1.62	0.00	1.27	0.92	1.14
Maximum waiting time in queue	8.16	3.56	2.97	3.24	0.00	3.59*	2.93*	3.63*
Average total time in system	6.44	5.10	4.16	6.71	4.26	5.33	1.19	1.48
Maximum total time in system	12.62	6.63	6.27	7.71	4.96	7.64*	2.95*	3.67*
Time-average number of parts in queue	0.79	0.18	0.36	0.16	0.05	0.31	0.29	0.36
Maximum number of parts in queue	3	1	2	1	1	1.60*	0.89*	1.11*
Drill-press utilization	0.92	0.59	0.90	0.51	0.70	0.72	0.18	0.23

*Taking means and standard deviations of the "maximum" measures might be of debatable validity—what is the "mean maximum" supposed to, well, mean? It might be better in these cases to take the *maximum* of the individual-replication maxima if you really want to know about the extremes.

The columns in Table 2-4 give the sample mean and sample standard deviation (see Appendix B) across the individual-replication results for the output performance measure in each row. The sample mean provides a more stable indication of what to expect from each performance measure than what happens on an individual replication, and the sample standard deviation measures cross-replication variation.

Since the individual replication results are independent and identically distributed, you could form a confidence interval for the true expected performance measure μ (think of μ as the sample mean across an infinite number of replications) as

$$\overline{X} \pm t_{n-1,1-\alpha/2} \frac{s}{\sqrt{n}}$$

where $\overline{X}$ is the sample mean, s is the sample standard deviation, n is the number of replications ($n = 5$ here), and $t_{n-1,1-\alpha/2}$ is the upper $1 - \alpha/2$ critical point from Student's t distribution with $n - 1$ degrees of freedom. Using the total-production measure, for example, this works out for a 95% confidence interval ($\alpha = 0.05$) to

$$3.80 \pm 2.776 \frac{1.64}{\sqrt{5}}$$

or 3.80 ± 2.04; the half widths for 95% confidence intervals on the expectations of all performance measures are in the last column of Table 2-4. The correct interpretation here is that in about 95% of the cases of making five simulation replications as we did, the interval formed like this will contain or "cover" the true (but unknown) expected value of total production. This confidence interval assumes that the total-production results from the replications are normally distributed, which is not really justified here, but we use this form anyway; see Section 6.3 for more on this issue of statistical *robustness*. You might notice that the half width of this interval (2.04) is pretty big compared to the value at its center (3.80); that is, the *precision* is not too good. This could be remedied by simply making more than the five replications we made, which looks like it was not enough to learn anything precise about the expected value of this output performance measure. The great thing about collecting your data by simulation is that you can always[2] go get more by simply calling for more replications.

2.6.3 Comparing Alternatives

Most simulation studies involve more than just a single setup or configuration of the system. People often want to see how changes in design, parameters (controllable in reality or not), or operation might affect performance. To see how randomness in simulation plays a role in these comparisons, we made a simple change to the example model and re-simulated (five replications).

The change we made was just to double the arrival rate—in other words, the mean interarrival time is now 2.5 minutes instead of 5 minutes. The exponential distribution is still used to generate interarrival times, and everything else in the simulation stays the same. This could represent, for instance, acquiring another customer for the facility, whose part-processing demands would be intermingled with the existing customer base.

[2] Well, almost always.

Figure 2-4 indicates what happened to the five "non-extreme" performance measures; the upper row (circles) of each plot indicates the original configuration (and is taken right out of Table 2-4), and the lower row (triangles) is for the new configuration. For each performance measure, the results of the five replications of each model are indicated, and the result from the first replication in each case is filled in (replications 2-5 are hollow).

It's not clear that total production really increases much with the double-time arrivals, due to limited processing capacity. It does appear, though, that the time in queue and in system, as well as the utilization, tend to increase, although not in every case (due to variability). And despite some overlap, the average queue lengths seem to grow somewhat when the arrival rate is doubled. While formal statistical analyses would be possible here to help sort things out (see Section 6.4), simple plots like these can sometimes be pretty illuminating (though you'd probably want substantially more than just five replications of each scenario).

Moreover, note that relying on just one replication (the first, indicated by the filled-in symbols) could be misleading. For instance, the average waiting time in queue in the first replications of each model variant would have suggested that it is far greater for the double-time arrivals, but looking at the spread across replications indicates that the difference isn't so clear-cut or dramatic. This is exactly the danger in relying on only a single run to make important decisions.

2.7 Simulating with Spreadsheets

You'll agree that simulating by hand doesn't have much of a future. People use spreadsheets for a lot of things, so how about simulation? Well, maybe, depending on the type and complexity of the model, and on what add-in software you might have. In this section, we'll discuss two examples, first a static model and then a dynamic model (recall Section 1.2.3); both, however, are very simple.

2.7.1 A News Vendor Problem

Rupert sells daily newspapers on a street corner. Each morning he must buy the same, fixed number q of copies from the printer at $c = 55$ cents each, and sells them for $r = \$1.00$ each through the day. He's noticed that demand D during a day is close to being a random variable X that's normally distributed with mean $\mu = 135.7$ and standard deviation $\sigma = 27.1$, except that D must be a nonnegative integer to make sense, so $D = \max(\lfloor X \rceil, 0)$ where $\lfloor \cdot \rceil$ rounds to the nearest integer (Rupert's not your average news vendor). Further, demands from day to day are independent of each other. Now if demand D in a day is no more than q, he can satisfy all customers and will have $q - D \geq 0$ papers left over, which he sells as scrap to the recycler on the next corner at the end of the day for $s = 3$ cents each (after all, it's old news at that point). But if $D > q$, he sells out all of his supply of q and just misses those other $D - q > 0$ sales. Each day starts afresh, independent of any other day, so this is a single-period (single-day) problem, and for a given day is a static model since it doesn't matter when individual customers show up through the day.

How many papers q should Rupert buy each morning to maximize his expected profit? If he buys too few, he'll sell out and miss out on the 45 cents profit on each lost

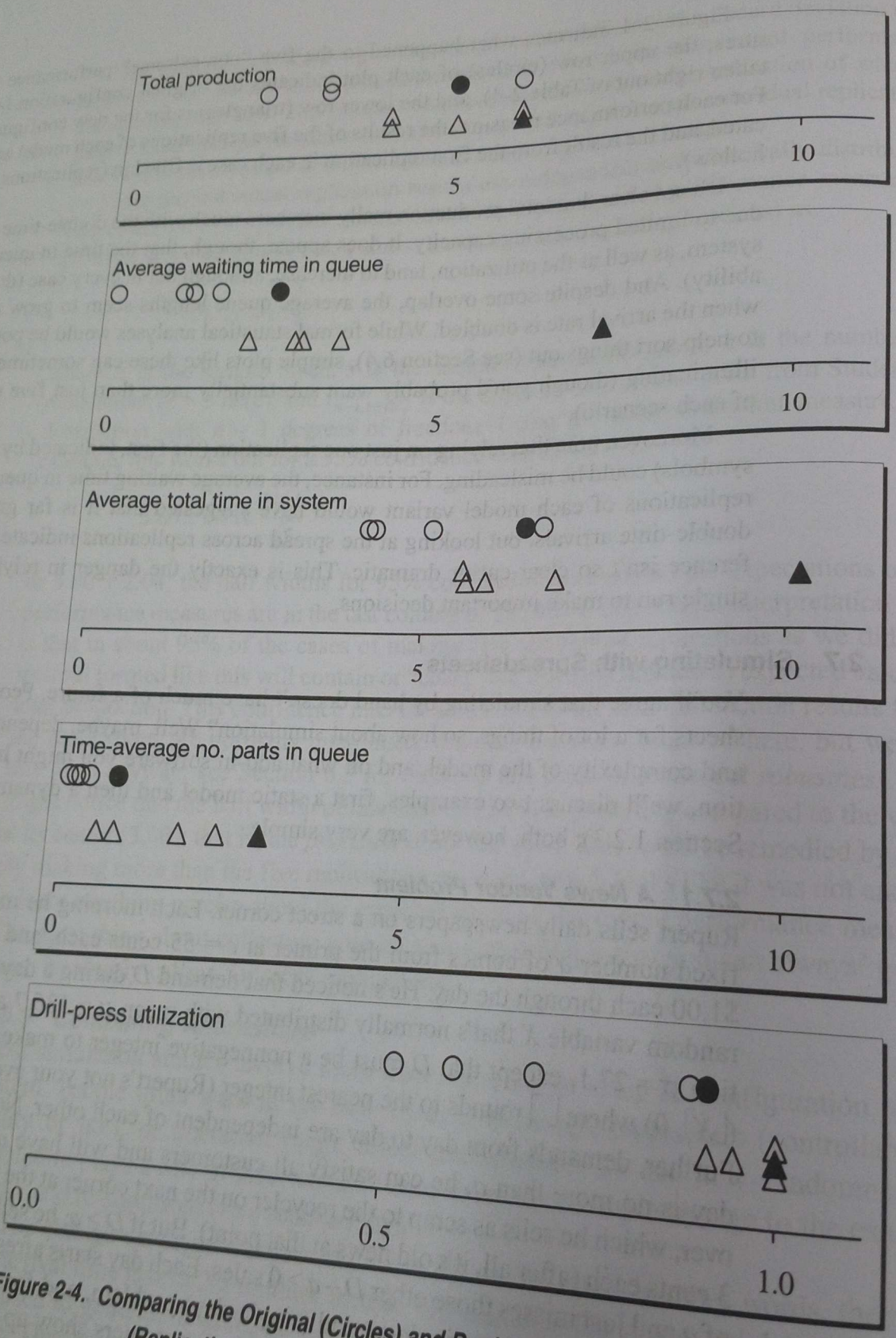


Figure 2-4. Comparing the Original (Circles) and Double-Time Arrival (Triangles) Systems (Replication 1 Is Filled In, and Replications 2-5 Are Hollow)

sale, but if he buys too many he will have some left over to scrap at a loss of 52 cents each. If he had a crystal ball and knew in the morning what demand was going to be that day, he'd of course buy just that many; Rupert's good, but he's not that good, so he needs to decide on a fixed value of q that will apply to every day.

This is a genuine classic in operations research, a single-period, perishable-inventory problem, with more serious applications like food stocks and blood banks. It has many versions, variants, and extensions, and a lot of attention has been paid to getting exact analytical solutions. But it's simple to simulate it; so simple, in fact, that it can be done in a spreadsheet.

First, let's figure out an expression for Rupert's profit on a typical day. Clearly, his daily cost is cq (we won't impose some enigmatic negative-good-will cost for lost sales if $D > q$ and he sells out). If it turns out that $D \le q$ then he sells D and meets all customer demand, gets revenue rD from customers, and has $q - D$ left over as scrap so gets $s(q - D)$ more revenue, such as it is, from the recycler. On the other hand, if $D > q$ he sells q, gets revenue rq from customers, but gets no scrap revenue. Thus, Rupert's total daily revenue is $r \min(D, q) + s \max(q - D, 0)$; to see this, just play out the two cases $D \le q$ and $D > q$. So his daily profit, in dollars, is

$$W(q) = r \min(D, q) + s \max(q - D, 0) - cq.$$

We've written this as a function of q since it clearly depends on Rupert's choice of q. In mathematical-programming or optimization parlance, q is Rupert's *decision variable* to try to maximize his *objective function* of expected daily profit, $\mathrm{E}(W(q))$. Since D is a random variable, so too is $W(q)$, and we need to take its expectation (see Appendix B for a refresher on probability and statistics) to get a deterministic objective function to try to maximize. Since expected values are long-run averages, in this case, over many days, Rupert is maximizing his long-run average daily profit by finding the best set-it-and-forget-it constant value of q. He's not trying to adjust q each day depending on demand, which is impossible since he has to commit to buying a fixed number q papers in the morning, before that day's demand is realized. Depending on the form of the probability distribution of D, it might be possible in some cases to derive an exact expression for $\mathrm{E}(W(q))$ that we could try to maximize with respect to q. But we can always build a simple simulation model and heed the lessons from Section 2.6 about statistical analysis of simulation output to get a valid and precise *estimate* of the optimal q, without making strong and maybe unrealistic assumptions about the distribution of D just to enable an analytical result.

To simulate this, we need to generate numerical values on the demand random variable $D = \max(\lfloor X \rceil, 0)$; such are called *random variates* on D. Thus, we first need to generate random variates on X, which has a normal distribution with mean $\mu = 135.7$ and standard deviation $\sigma = 27.1$. Section 12.2 discusses random-variate generation in general, so we'll trust you to look ahead if you're interested at this point. In this case we can generate a normally distributed random variate as $X = \Phi^{-1}_{\mu,\sigma}(U)$ where U is a *random number* (continuous random variate distributed uniformly between 0 and 1, discussed in Section 12.1), $\Phi_{\mu,\sigma}$ is the cumulative distribution of the normal

distribution with mean μ and standard deviation σ, and $\Phi^{-1}_{\mu,\sigma}$ is the functional inverse of $\Phi_{\mu,\sigma}$, i.e., $\Phi^{-1}_{\mu,\sigma}$ "undoes" whatever $\Phi_{\mu,\sigma}$ does. Another way of looking at the functional inverse is that we need to solve $U = \Phi_{\mu,\sigma}(X)$ for X, where U would be a specific number between 0 and 1 from the random-number generator. As you may know, neither $\Phi_{\mu,\sigma}$ nor $\Phi^{-1}_{\mu,\sigma}$ for the normal distribution can be written as an explicit closed-form formula, so evaluation of $\Phi^{-1}_{\mu,\sigma}(U)$ must be by numerical approximation, but this is built into spreadsheet software, and certainly into simulation software as well.

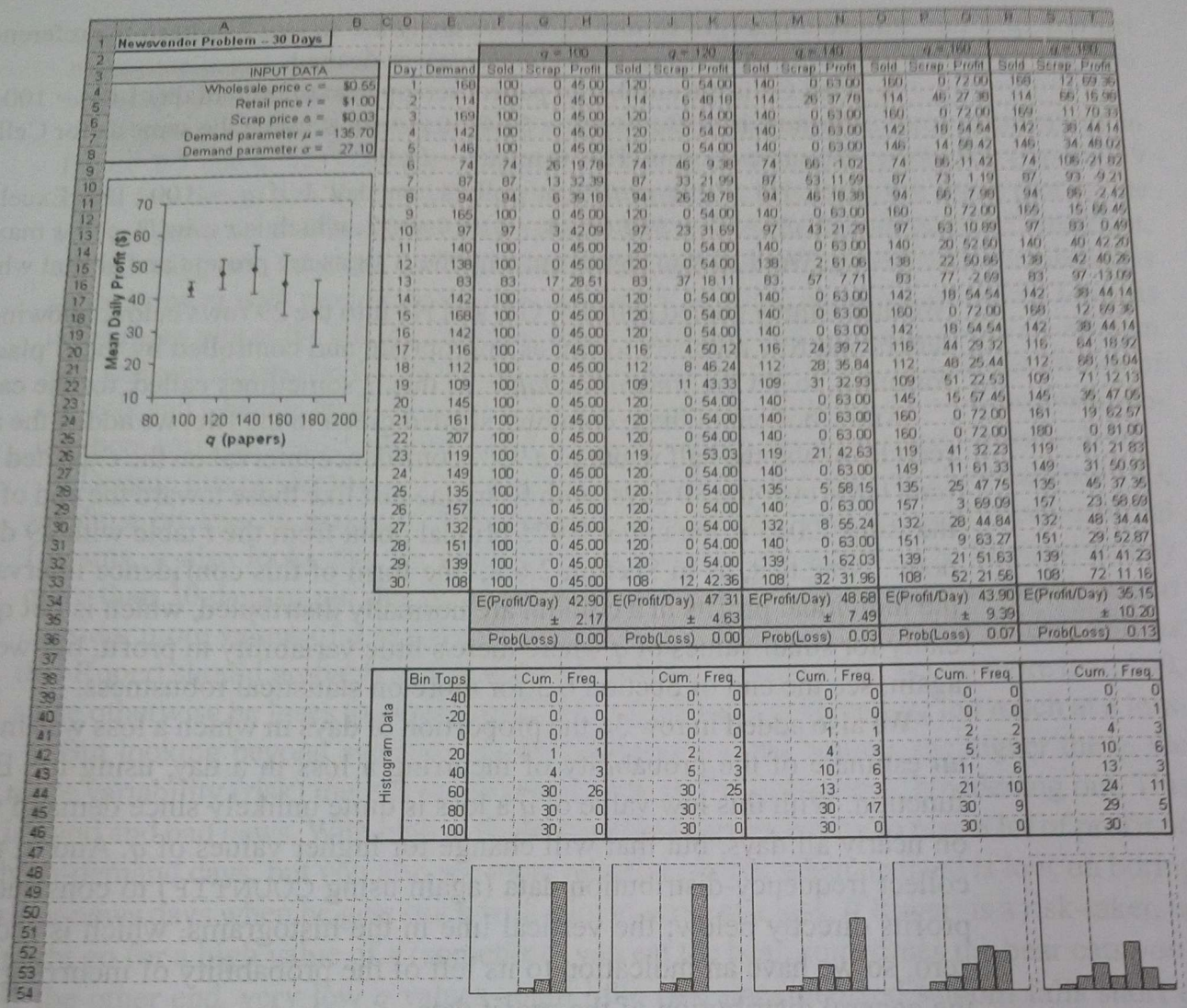
We used the Microsoft® Excel spreadsheet to create the file `Newsvendor.xls` in Figure 2-5. You'll find this file in the Book Examples folder, which is in turn in the Arena folder if you followed the instructions in Appendix D for Arena installation from the CD supplied with this book; a typical complete path to this folder would be `C:\Program Files\Rockwell Software\Arena\Book Examples`. In cells B4 – B8 (shaded blue in the file) are the input parameters, and in row 2 (pink) are some trial values for the decision variable q (cells H2, K2, N2, Q2, and T2). Column D just gives the day number for reference. We simulated 30 days, though as mentioned earlier, each day is independent so is really an independent, static simulation.

Column E has the demand on each day, via the Excel formula

```
MAX(ROUND(NORMINV(RAND(), $B$7, $B$8), 0),0)
```

that is the same all the way down the column. Let's dissect this formula from the inside out:

- `RAND()` is Excel's built-in random-number generator, which returns a number between 0 and 1 that's supposed to be uniformly distributed and independent across draws. See Section 12.1 for a discussion of random-number generators, which are more subtle than you might think, but are clearly essential to good stochastic simulation. Unfortunately, not all random-number generators supplied with various software packages are of high quality (however, the one built into Arena is excellent). Excel's `RAND()` function is "volatile" by default, meaning that every instance of it re-draws a "fresh" random number upon any recalculation of the spreadsheet, which you can force by hitting the F9 key, by saving the file, or by editing just about anything anywhere in the sheet.
- `NORMINV`(u, μ, σ) is Excel's built-in numerical approximation to $\Phi^{-1}_{\mu,\sigma}$, so when used with u = `RAND()` returns a random variate distributed normally with mean μ and standard deviation σ. We have μ in cell B7 and σ in cell B8; the $ in Excel-formula cell references forces what immediately follows it (column letter or row number) not to change when the formula is copied into other cells, and here we want both the column and row to stay the same when this formula is copied into other cells since μ and σ are in these fixed cells.
- `ROUND`$(x, 0)$ in Excel returns x rounded to the nearest integer. The second argument's being 0 ensures that the rounding is to the nearest integer as opposed to other kinds of rounding (see Excel's Help).
- `MAX`$(a, b, \ldots)$, as you might guess, returns the maximum of its arguments.



Newsvendor Problem – 30 Days

INPUT DATA

Wholesale price c =	$0.55
Retail price r =	$1.00
Scrap price a =	$0.03
Demand parameter μ =	135.70
Demand parameter σ =	27.10

Mean Daily Profit ($) vs q (papers)

Day	Demand	q = 100 Sold	Scrap	Profit	q = 120 Sold	Scrap	Profit	q = 140 Sold	Scrap	Profit	q = 160 Sold	Scrap	Profit	q = 180 Sold	Scrap	Profit
1	160	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	168	12	69.36
2	114	100	0	45.00	114	6	40.18	114	26	37.78	114	46	27.38	114	66	16.98
3	169	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	169	11	70.33
4	142	100	0	45.00	120	0	54.00	140	0	63.00	142	18	54.54	142	38	44.14
5	146	100	0	45.00	120	0	54.00	140	0	63.00	146	14	58.42	146	34	48.02
6	74	74	26	19.78	74	46	9.38	74	66	-1.02	74	86	-11.42	74	106	-21.82
7	87	87	13	32.39	87	33	21.99	87	53	11.59	87	73	1.19	87	93	-9.21
8	94	94	6	39.18	94	26	28.78	94	46	18.38	94	66	7.98	94	86	-2.42
9	165	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	165	15	66.45
10	97	97	3	42.09	97	23	31.69	97	43	21.29	97	63	10.89	97	83	0.49
11	140	100	0	45.00	120	0	54.00	140	0	63.00	140	20	52.60	140	40	42.20
12	138	100	0	45.00	120	0	54.00	138	2	61.06	138	22	50.66	138	42	40.26
13	83	83	17	28.51	83	37	18.11	83	57	7.71	83	77	-2.69	83	97	-13.09
14	142	100	0	45.00	120	0	54.00	140	0	63.00	142	18	54.54	142	38	44.14
15	168	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	168	12	69.36
16	142	100	0	45.00	120	0	54.00	140	0	63.00	142	18	54.54	142	38	44.14
17	116	100	0	45.00	116	4	50.12	116	24	39.72	116	44	29.32	116	84	18.92
18	112	100	0	45.00	112	8	46.24	112	28	35.84	112	48	25.44	112	88	15.04
19	109	100	0	45.00	109	11	43.33	109	31	32.93	109	51	22.53	109	71	12.13
20	145	100	0	45.00	120	0	54.00	140	0	63.00	145	15	57.45	145	35	47.05
21	161	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	161	19	62.57
22	207	100	0	45.00	120	0	54.00	140	0	63.00	160	0	72.00	180	0	81.00
23	119	100	0	45.00	119	1	53.03	119	21	42.63	119	41	32.23	119	61	21.83
24	149	100	0	45.00	120	0	54.00	140	0	63.00	149	11	61.33	149	31	50.93
25	135	100	0	45.00	120	0	54.00	135	5	58.15	135	25	47.75	135	45	37.35
26	157	100	0	45.00	120	0	54.00	140	0	63.00	157	3	69.09	157	23	58.69
27	132	100	0	45.00	120	0	54.00	132	8	55.24	132	28	44.84	132	48	34.44
28	151	100	0	45.00	120	0	54.00	140	0	63.00	151	9	63.27	151	29	52.87
29	139	100	0	45.00	120	0	54.00	139	1	62.03	139	21	51.63	139	41	41.23
30	108	100	0	45.00	108	12	42.36	108	32	31.96	108	52	21.56	108	72	11.16
		E(Profit/Day)		42.90	E(Profit/Day)		47.31	E(Profit/Day)		48.68	E(Profit/Day)		43.90	E(Profit/Day)		35.15
			±	2.17		±	4.63		±	7.49		±	9.39		±	10.20
		Prob(Loss)		0.00	Prob(Loss)		0.00	Prob(Loss)		0.03	Prob(Loss)		0.07	Prob(Loss)		0.13

Histogram Data

Bin Tops	Cum.	Freq.	Cum.	Freq.	Cum.	Freq.	Cum.	Freq.	Cum.	Freq.
-40	0	0	0	0	0	0	0	0	0	0
-20	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	2	2	4	3
20	1	1	2	2	4	3	5	3	10	6
40	4	3	5	3	10	6	11	6	13	3
60	30	26	30	25	13	3	21	10	24	11
80	30	0	30	0	30	17	30	9	29	5
100	30	0	30	0	30	0	30	0	30	1

Figure 2-5. Spreadsheet Simulation of the News Vendor Problem

We entered this formula originally in cell E4, then just copied it to the 29 cells below to get independent demands on 30 days, as `RAND()` draws a fresh random number each time it appears. Note that if you hit the F9 key or otherwise force the sheet to recalculate, you get a new column of demand numbers, and most other things in the sheet also change accordingly.

For the case of $q = 100$, columns F, G, and H in rows 4 – 33 contain the number of papers sold, scrapped, and the daily profit, for each of days 1 – 30:

- Cell F4 is the number sold on day 1 if $q = 100$. The Excel formula is = `MIN($E4, H$2)`. `$E4` is the demand for day 1. We apply the `$` to column E since we'll later want to copy this formula for other values of q in columns I, L, O, and R but keep the column reference to E, but no `$` for row 4 since we want it to change to rows 5 – 33 to pick up demands for subsequent days. `H$2` is $q = 100$. We have no `$` for column H since we'll later want it to change for other q values in

columns I, L, O, and R, but we apply a $ to row 2 to keep the reference to it for rows below. Thus, this is $\min(D, q)$, as desired.

- Cell G4 is the number of papers scrapped at the end of day 1 if $q = 100$. The Excel formula is `=MAX(H$2 - $E4, 0)`, with $ usage the same as for Cell F4 and for the same reasons. This is $\max(q - D, 0)$.
- Cell H4 is the profit, in dollars, on day 1 if $q = 100$. The Excel formula is `=$B$5*F4 + $B$6*G4 - $B$4*H$2`, which is $r \min(D, q) + s \max(q - D, 0) - cq$. We'll let you figure out why the $ signs are present and absent where they are.

We then simply copied cells F4, G4, and H4 into the 29 rows below, allowing the column letters and row numbers to run as appropriate and controlled by the $ placement in the formulas, to get the *simulation table*, as this is sometimes called, for the case $q = 100$.

At the bottom of these columns, shaded green in the file, we added the average profit (cell H34) and the half width of a 95% confidence interval on the expected average profit (cell H35) over the 30 days, with formulae just like those toward the end of Section 2.6.2; the value 2.045 is the upper 0.975 critical point from the t table with 29 degrees of freedom[3]. Note that, as in Section 2.6.2, the form of this confidence interval assumes that the individual profits in a column are normally distributed, which is not quite true, especially for small values of q where there's little variability in profit, but we do so anyway; again, see the end of Section 6.3 for more on statistical robustness.

We also added in row 36 the proportion of days in which a loss was incurred, which is an estimate of the probability of incurring a loss in a day, using the Excel `COUNTIF` function. With this low value of q a loss is quite unlikely since demand will be at least q on nearly all days, but that will change for higher values of q. And in rows 38 – 46 we collect frequency-distribution data (again using `COUNTIF`) to construct a histogram of profits directly below; the vertical line in the histograms, which is red in the file, is at zero, so we have an indication to its left of the probability of incurring a loss, as well as the general distribution of the profit data.

To complete the simulation study, we took columns F, G, and H and replicated them into four more blocks of three columns each out to the right, but with q taking the values 120, 140, 160, and 180, and added histograms at the bottom for each value of q. To visualize the mean results, we added the plot on the left edge to show the average profit as a dot, with vertical I-beams showing the confidence intervals, for different values of q on the horizontal axis.

Notice that, for a given day, we chose to use the same realized numerical demand variate (column E) for all trial values of q, rather than generate fresh independent demands for each value of q, which would have been valid as well. We're not lazy (at least about this) and filling more Excel columns with fresh demands wouldn't cost much, but we did it this way deliberately. Our interest is really in *comparing* profit for different values of q, and by using the same realized numerical demands on a given day for all q, the differences we see in profit will be attributable to differences in q (which is the effect we're trying to measure) rather than to chance differences in demand. This is an example of a *variance-reduction technique*, in particular of *common random numbers*, discussed

[3] More generally, we could use TINV(0.05, COUNT(H4:H33) - 1) instead of hard-wiring in 2.045.

more fully in Section 12.4. Such strategies in stochastic simulation can help you get more precise answers, often with little or no more work (in this case, with *less* work) and are worth thinking about.

If you open the spreadsheet file yourself, you won't see the same numbers as in Figure 2-5 due to the volatility of Excel's RAND( ). And as you recalculate (tap the F9 key a few times) you'll see all the numbers change, the objects in the mean plot on the left will jump up and down, and the histograms at the bottom will undulate left and right, all reflecting this volatility and the underlying truth that such a stochastic simulation is an experiment with random outcomes; indeed the jumping up and down of the I-beams in the mean plot at the left is a reminder that confidence intervals themselves are random intervals and depend on the particular sample you happened to get. Notice that all five of the dots and I-beams tend to jump up or down together, reflecting the common-random-numbers variance reduction mentioned in the preceding paragraph; see Exercise 2-10.

Despite this randomness, it looks like Rupert should probably set q to be around 140, maybe a bit less, to maximize his expected daily profit. From the confidence-interval I-beams, it also looks like we might want to increase the sample size to considerably more than 30 days to pin this down better (Exercise 2-11). And clearly it would help to consider more values of q in a finer mesh (Exercise 2-12), but this study at least tells us that Rupert should probably buy at least 120 papers each morning, but no more than 160, since otherwise he loses too much profit due to lost sales or scrapping too much at a loss.

But looking beyond averages via the histograms at the bottom, the higher the q, the more variability (risk) there is in the profit, seen by the histograms' broadening out. This is good and bad news. With high q, Rupert has the large inventory to reap a lot of profit on high-demand days, but you also see more negative profits when demand is low, on boring slow-news days when he gets stuck with a lot to scrap at a loss. If Rupert is a risk-taker, he might go for a high value of q (sometimes you eat the bear, sometimes the bear eats you). At the other end, very low q values entail little risk since demand seldom falls short of such small q's, but the downside is that average profit is also low as Rupert cannot cash in on high-demand days since he sells out. Such risk/return tradeoffs in business decisions are commonly quantified via spreadsheet simulations of static models like this.

It's quite easy to simulate the news vendor problem in Arena; see Exercises 6.22 and 6.23.

2.7.2 A Single-Server Queue

Spreadsheets might also be used to simulate some dynamic models, but only very simple ones. Consider a single-server queue that's logically the same as what we plodded through by hand in Section 2.4, but different on some specifics. As before, customers arrive one at a time (the first customer arrives at time 0 to find the system empty and idle), may have to wait in a FIFO queue, are served one at a time by a single server, and then depart. But we'll make the following changes from the hand-simulated model:

- Interarrival times have an exponential probability distribution with mean $1/\lambda = 1.6$ minutes, and are independent of each other (see Appendices B and C). The parameter $\lambda = 1/1.6 = 0.625$ is the *arrival rate* (per minute here).

- Service times have a continuous uniform distribution between $a = 0.27$ minute and $b = 2.29$ minutes, are independent of each other, and are independent of the arrival times.

This is called the *M/U/1 queue*, where the U refers to the uniformly distributed service times.

Instead of observing all the outputs in the hand simulation, we'll just watch the waiting times in queue, $WQ_1, WQ_2, \ldots, WQ_{50}$ of the first 50 customers to complete their queue waits (not including their service times). In the hand simulation, we set up a clock, state variables, an event calendar, and statistical accumulators to simulate this, but *for a single-server FIFO queue* there is a simple recursion for the WQ_i values (if that's all we're interested in by way of output), Lindley's (1952) formula. Let S_i be the time in service for customer i, and let A_i be the interarrival time between customers $i-1$ and i. Then

$$WQ_i = \max(WQ_{i-1} + S_{i-1} - A_i, 0) \quad \text{for } i = 2, 3, \ldots$$

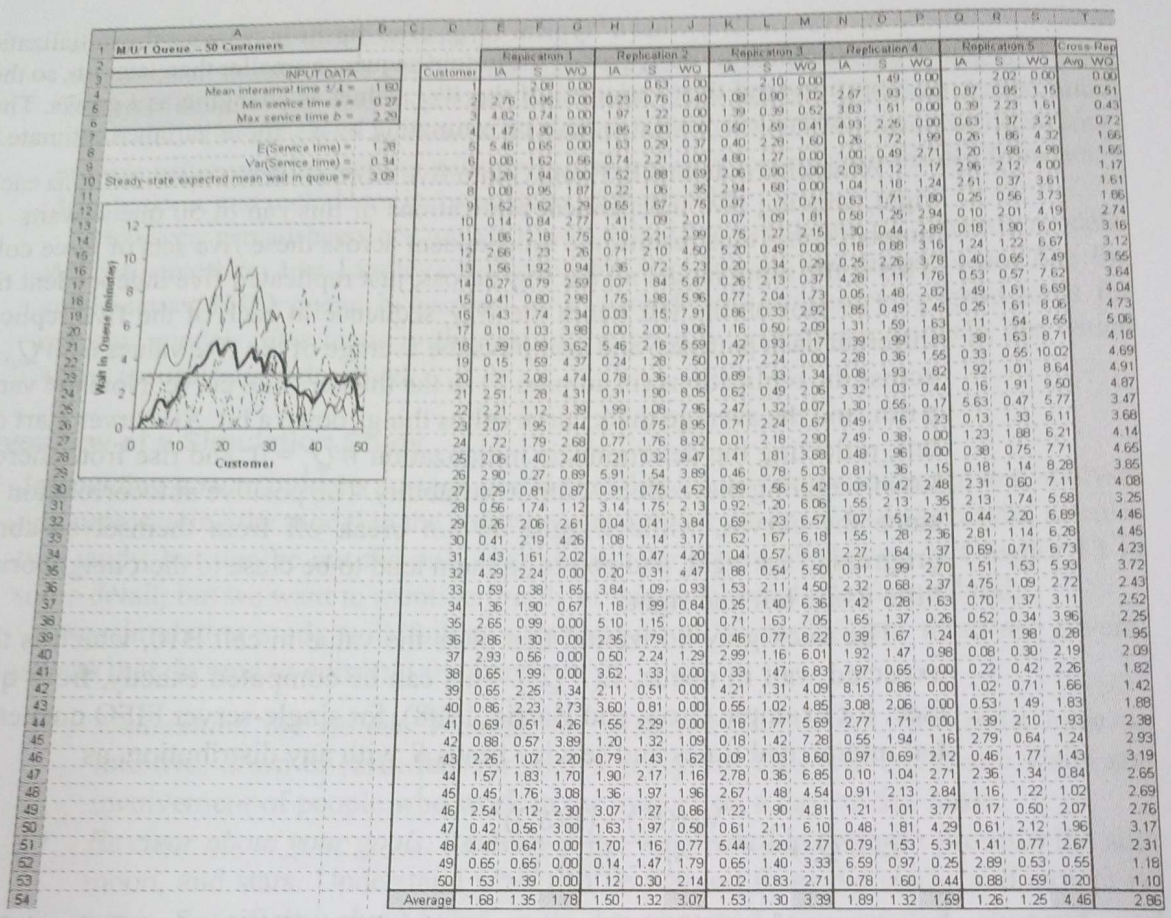
and we initialize by setting $WQ_1 = 0$ since the first customer arrives to an empty and idle system.

To simulate this, we need to generate random variates from the exponential and continuous uniform distributions. Again referring ahead to Section 12.2, there are simple formulas for both (as before, U denotes a random number distributed continuously uniformly between 0 and 1):

- Exponential interarrival times with mean $1/\lambda$: $A_i = -(1/\lambda) \ln(1 - U)$ where ln is the natural (base e) logarithm.
- Continuously uniformly distributed service times between a and b: $S_i = a + (b - a)U$.

Figure 2-6 shows the Excel file `MU1.xls` in the Book Examples folder. Cells B4, B5, and B6 are the parameters, and column D just gives the customer number $i = 1, 2, \ldots, 50$ for a run of the model. Interarrival times are in column E, using the Excel formula `-$B$4*LN(1 - RAND())`, and service times are in column F, generated by the formula `$B$5 + ($B$6 - $B$5)*RAND()`.

Lindley's recursion is in column G, initialized by the 0 in cell G4. A typical entry is in cell G9, `MAX(G8 + F8 - E9, 0)`, which generates WQ_6 from WQ_5 (in cell G8), S_5 (cell F8), and A_6 (cell E9). In this way, each queue wait in column G is linked to the one above it, establishing a correlation between them. Since this is correlation of entries in a sequence with each other, it's called *autocorrelation*. In this case, it's *positive* autocorrelation since a large wait probably means that the next wait will be large too, and vice-versa, which you can understand from looking at Lindley's recursion or just from your own painful experiences standing in line. So, unlike the news vendor's profit rows in Section 2.7.1, the waiting times in queue here are *not* independent from row to row in column G, which is typical of the output sequence within a run of a dynamic simulation.



M/U/1 Queue – 50 Customers

INPUT DATA

Mean interarrival time 1/λ =	1.60
Min service time a =	0.27
Max service time b =	2.29
E(Service time) =	1.28
Var(Service time) =	0.34
Steady-state expected mean wait in queue =	3.09

Customer	Replication 1			Replication 2			Replication 3			Replication 4			Replication 5			Cross-Rep
	IA	S	WQ	IA	S	WQ	IA	S	WQ	IA	S	WQ	IA	S	WQ	Avg WQ
1		2.09	0.00		0.63	0.00		2.10	0.00		1.49	0.00		2.02	0.00	0.00
2	2.76	0.95	0.00	0.23	0.76	0.40	1.08	0.90	1.02	2.67	1.93	0.00	0.87	0.85	1.15	0.51
3	4.82	0.74	0.00	1.97	1.22	0.00	1.39	0.39	0.52	3.63	1.51	0.00	0.39	2.23	1.61	0.43
4	1.47	1.90	0.00	1.86	2.00	0.00	0.50	1.59	0.41	4.93	2.25	0.00	0.63	1.37	3.21	0.72
5	5.46	0.65	0.00	1.63	0.29	0.37	0.40	2.29	1.60	0.26	1.72	1.99	0.26	1.86	4.32	1.66
6	0.00	1.62	0.56	0.74	2.21	0.00	4.80	1.27	0.00	1.00	0.49	2.71	1.20	1.98	4.98	1.65
7	3.29	1.94	0.00	1.52	0.88	0.69	2.06	0.90	0.00	2.03	1.12	1.17	2.96	2.12	4.00	1.17
8	0.00	0.95	1.87	0.22	1.06	1.36	2.94	1.68	0.00	1.04	1.16	1.24	2.51	0.37	3.61	1.61
9	1.52	1.62	1.29	0.65	1.67	1.76	0.97	1.17	0.71	0.63	1.71	1.80	0.25	0.56	3.73	1.86
10	0.14	0.84	2.77	1.41	1.09	2.01	0.07	1.09	1.81	0.58	1.25	2.94	0.10	2.01	4.19	2.74
11	1.86	2.18	1.75	0.10	2.21	2.99	0.75	1.27	2.15	1.30	0.44	2.89	0.18	1.90	6.01	3.16
12	2.66	1.23	1.26	0.71	2.10	4.50	5.20	0.49	0.00	0.18	0.68	3.16	1.24	1.22	6.67	3.12
13	1.56	1.92	0.94	1.36	0.72	5.23	0.20	0.34	0.29	0.25	2.26	3.78	0.40	0.65	7.49	3.55
14	0.27	0.79	2.59	0.07	1.84	5.87	0.26	2.13	0.37	4.28	1.11	1.76	0.53	0.57	7.62	3.64
15	0.41	0.80	2.98	1.75	1.98	5.96	0.77	2.04	1.73	0.05	1.48	2.82	1.49	1.61	6.69	4.04
16	1.43	1.74	2.34	0.03	1.15	7.91	0.86	0.33	2.92	1.85	0.49	2.45	0.25	1.61	8.06	4.73
17	0.10	1.03	3.98	0.00	2.00	9.06	1.16	0.50	2.09	1.31	1.59	1.63	1.11	1.54	8.55	5.06
18	1.39	0.89	3.62	5.46	2.16	5.59	1.42	0.93	1.17	1.39	1.99	1.83	1.38	1.63	8.71	4.18
19	0.15	1.59	4.37	0.24	1.28	7.50	10.27	2.24	0.00	2.28	0.36	1.55	0.33	0.55	10.02	4.69
20	1.21	2.08	4.74	0.78	0.36	8.00	0.89	1.33	1.34	0.08	1.93	1.82	1.92	1.01	8.64	4.91
21	2.51	1.28	4.31	0.31	1.90	8.05	0.62	0.49	2.06	3.32	1.03	0.44	0.16	1.91	9.50	4.87
22	2.21	1.12	3.39	1.99	1.08	7.96	2.47	1.32	0.07	1.30	0.56	0.17	5.63	0.47	5.77	3.47
23	2.07	1.95	2.44	0.10	0.75	8.95	0.71	2.24	0.67	0.49	1.16	0.23	0.13	1.33	6.11	3.68
24	1.72	1.79	2.68	0.77	1.76	8.92	0.01	2.18	2.90	7.49	0.38	0.00	1.23	1.88	6.21	4.14
25	2.06	1.40	2.40	1.21	0.32	9.47	1.41	1.81	3.68	0.48	1.96	0.00	0.38	0.75	7.71	4.66
26	2.90	0.27	0.89	5.91	1.54	3.89	0.46	0.78	5.03	0.81	1.36	1.15	0.18	1.14	8.28	3.85
27	0.29	0.81	0.87	0.91	0.75	4.52	0.39	1.56	5.42	0.03	0.42	2.48	2.31	0.60	7.11	4.08
28	0.56	1.74	1.12	3.14	1.75	2.13	0.92	1.20	6.06	1.55	2.13	1.35	2.13	1.74	5.58	3.25
29	0.26	2.06	2.61	0.04	0.41	3.84	0.69	1.23	6.57	1.07	1.51	2.41	0.44	2.20	6.89	4.46
30	0.41	2.19	4.26	1.08	1.14	3.17	1.62	1.67	6.18	1.55	1.28	2.36	2.81	1.14	6.28	4.46
31	4.43	1.61	2.02	0.11	0.47	4.20	1.04	0.57	6.81	2.27	1.64	1.37	0.69	0.71	6.73	4.23
32	4.29	2.24	0.00	0.20	0.31	4.47	1.88	0.54	5.50	0.31	1.99	2.70	1.51	1.53	5.93	3.72
33	0.59	0.38	1.65	3.84	1.09	0.93	1.53	2.11	4.50	2.32	0.68	2.37	4.75	1.09	2.72	2.43
34	1.36	1.90	0.67	1.18	1.99	0.84	0.25	1.40	6.36	1.42	0.28	1.63	0.70	1.37	3.11	2.52
35	2.65	0.99	0.00	5.10	1.15	0.00	0.71	1.63	7.05	1.65	1.37	0.26	0.52	0.34	3.96	2.25
36	2.86	1.30	0.00	2.35	1.79	0.00	0.46	0.77	8.22	0.39	1.67	1.24	4.01	1.98	0.28	1.95
37	2.93	0.56	0.00	0.50	2.24	1.29	2.99	1.16	6.01	1.92	1.47	0.98	0.08	0.30	2.19	2.09
38	0.65	1.99	0.00	3.62	1.33	0.00	0.33	1.47	6.83	7.97	0.65	0.00	0.22	0.42	2.26	1.82
39	0.65	2.25	1.34	2.11	0.51	0.00	4.21	1.31	4.09	8.15	0.86	0.00	1.02	0.71	1.66	1.42
40	0.86	2.23	2.73	3.60	0.81	0.00	0.55	1.02	4.85	3.08	2.06	0.00	0.53	1.49	1.83	1.88
41	0.69	0.51	4.26	1.55	2.29	0.00	0.18	1.77	5.69	2.77	1.71	0.00	1.39	2.10	1.93	2.38
42	0.88	0.57	3.89	1.20	1.32	1.09	0.18	1.42	7.28	0.55	1.94	1.16	2.79	0.64	1.24	2.93
43	2.26	1.07	2.20	0.79	1.43	1.62	0.09	1.03	8.60	0.97	0.69	2.12	0.46	1.77	1.43	3.19
44	1.57	1.83	1.70	1.90	2.17	1.16	2.78	0.36	6.85	0.10	1.04	2.71	2.36	1.34	0.84	2.65
45	0.45	1.76	3.08	1.36	1.97	1.96	2.67	1.48	4.54	0.91	2.13	2.84	1.16	1.22	1.02	2.69
46	2.54	1.12	2.30	3.07	1.27	0.86	1.22	1.90	4.81	1.21	1.01	3.77	0.17	0.50	2.07	2.76
47	0.42	0.56	3.00	1.63	1.97	0.50	0.61	2.11	6.10	0.48	1.81	4.29	0.61	2.12	1.96	3.17
48	4.40	0.64	0.00	1.70	1.16	0.77	5.44	1.20	2.77	0.79	1.53	5.31	1.41	0.77	2.67	2.31
49	0.65	0.65	0.00	0.14	1.47	1.79	0.65	1.40	3.33	6.59	0.97	0.25	2.89	0.53	0.55	1.18
50	1.53	1.39	0.00	1.12	0.30	2.14	2.02	0.83	2.71	0.78	1.60	0.44	0.88	0.59	0.20	1.10
Average	1.68	1.35	1.78	1.50	1.32	3.07	1.53	1.30	3.39	1.89	1.32	1.59	1.26	1.25	4.46	2.86

Figure 2-6. Spreadsheet Simulation of the M/U/1 Queue

At the bottom of columns E, F, and G we give the averages. For columns E and F, this is just to confirm that they're close to the expected values of the interarrival times and service times, respectively $1/\lambda = 1.6$ and $(a + b)/2 = 1.28$ (see Appendix C). For column G, this is the average wait in queue for the first 50 customers; i.e., the output result of the simulation.

We could also easily compute the sample standard deviation of the queue waits in column G, and then confidence intervals as in the news vendor simulation, but we refrain from doing so for two very important reasons:

1. Due to the autocorrelation among the WQ_i values in column G, the sample variance would be *biased* as an estimator of the variance of a "typical" WQ_i, since they are to some extent "pegged" to each other so are not free to vary as much as they would if they were independent. With our positive autocorrelation, the sample variance estimator would be biased low, i.e., would be too small and understate the variance of a "typical" WQ_i. Just because you're able to compute some number, or for that matter plot some graph, you shouldn't if it could be misleading.

2. It's not clear what a "typical" WQ_i even means here, since the initialization $WQ_1 = 0$ makes the early part of the output sequence smaller than later on, so the distribution (and thus variance) of the WQ_i values is changing as i grows. Thus, it's not even clear what it is we'd be estimating by a sample-variance estimate.

We copied columns E, F, and G into five more blocks of three columns each out to the right, creating five independent replications of this run of 50 queue waits. All random numbers, and thus results, are independent across these five sets of three columns each. This is the same model all the way across, just replicated five independent times.

The plot on the left traces the WQ_i sequence in each of the five replications using different dash patterns, and also plots the average of the five values of WQ_i, for a fixed i, across the replications (in column T) as the thicker solid curve. Note the variation within a run, and the average thick curve calms things down a bit. All curves start off at 0 on the left, reflecting the deterministic initialization $WQ_1 = 0$, and rise from there, subject to a lot of wiggling that reflects output variability. The positive autocorrelation is in evidence since the curves, while wiggly, do not break off from themselves abruptly as they progress to the right, and points on them tend to be close to their neighboring points for a little while left and right.

The thick gray horizontal line plots the value in cell B10, which is the steady-state expected wait in queue, $E(WQ_\infty)$. This can be computed exactly, from queueing theory (see, for example Gross and Harris, 1998), for single-server FIFO queues with exponential interarrival times and service times S_i with any distribution, as

$$E(WQ_\infty) = \lambda \frac{\text{Var}(S_i) + (E(S_i))^2}{2(1 - \lambda E(S_i))} .$$

In our case, $\lambda = 0.625$ (the reciprocal of cell B4), $E(S_i) = (a + b)/2 = 1.28$ in cell B8, and $\text{Var}(S_i) = (b - a)^2/12 = 0.34$ in cell B9 (see Appendix C). Though the variation in the plot obscures it, a little imagination might suggest that the thick solid average curve is converging to the gray horizontal line; see Exercise 2-13.

Why no histograms like those at the bottom in news vendor problem spreadsheet? They could have been constructed in the same way, and could provide useful information about the distribution of the queue waits, beyond just their means. The potential problem with this is the autocorrelation in the queue-wait sequence, so a histogram, like a variance estimate, could be biased if our run were not long enough to ensure that the waiting times in queue eventually visit all possible regions and in the right proportions; the autocorrelation links together nearby observations, which would impede such visitation if the run were too short. However, a very long run could indeed produce a nearly unbiased histogram, which could be useful; in practice, it might be hard to determine how long is long enough. The question is whether to present something that's accurate and useful most of the time, at the risk of its being occasionally biased and misleading.

Simulating a single-server queue in Arena is quite simple, and is basically the central example we use in Chapter 3 to introduce Arena.

2.7.3 Extensions and Limitations

Spreadsheet simulation is popular for static models, many involving financial or risk analysis. Commercial add-in packages to Excel, like @RISK (Palisade Corporation, 2009) and Crystal Ball® (Oracle Corporation, 2009), facilitate common operations, provide better random-number generators, make it easy to generate variates from many distributions, and include tools for analysis of the results.

However, spreadsheets are not very well suited for simulation of dynamic models. Absent something like Lindley's recursion for very special and simple cases like the single-server FIFO queue in Section 2.7.2, spreadsheets are not very good tools for dynamic models. Our focus in the rest of the book is on dynamic models, for which Arena is specifically designed.

2.8 Overview of a Simulation Study

In deciding how to model a system, you'll find that issues related to design and analysis and representing the model in the software certainly are essential to a successful simulation study, but they're not the only ingredients. We'll take all this up in Chapter 13 in some detail, but we want to mention briefly at this early point what's involved.

No simulation study will follow a cut-and-dried "formula," but there are several aspects that do tend to come up frequently:

- *Understand the system*. Whether the system exists or not, you must have an intuitive, down-to-earth feel for what's going on. This will entail site visits and involvement of people who work in the system on a day-to-day basis.
- *Be clear about your goals*. Realism is the watchword here; don't promise the sun, moon, and stars. Understand what can be learned from the study, and expect no more. Specificity about what is to be observed, manipulated, changed, and delivered is essential. And return to these goals throughout the simulation study to keep your attention focused on what's important, namely, making decisions about how best (or at least better) to operate the system.
- *Formulate the model representation*. What level of detail is appropriate? What needs to be modeled carefully and what can be dealt with in a fairly crude, high-level manner? Get buy-ins to the modeling assumptions from management and those in decision-making positions.
- *Translate into modeling software*. Once the modeling assumptions are agreed upon, represent them faithfully in the simulation software. If there are difficulties, be sure to iron them out in an open and honest way rather than burying them. Involve those who really know what's going on (animation can be a big help here).
- *Verify that your computer representation represents the conceptual model faithfully*. Probe the extreme regions of the input parameters, verify that the right things happen with "obvious" input, and walk through the logic with those familiar with the system.
- *Validate the model*. Do the input distributions match what you've observed in the field? Do the output performance measures from the model match up with those

from reality? While statistical tests can be carried out here, a good dose of common sense is also valuable.

- *Design the experiments*. Plan out what it is you want to know and how your simulation experiments will get you to the answers in a precise and efficient way. Often, principles of classical statistical experimental design can be of great help here.
- *Run the experiments*. This is where you go to lunch while the computer is grinding merrily away, or maybe go home for the night or the weekend, or go on vacation. The need for careful experimental design here is clear. But don't panic—your computer probably spends most of its time doing nothing, so carrying out your erroneous instructions doesn't constitute the end of the world (remember, you're going to make your mistakes on the computer where they don't count rather than for real where they do).
- *Analyze your results*. Carry out the right kinds of statistical analyses to be able to make accurate and precise statements. This is clearly tied up intimately with the design of the simulation experiments.
- *Get insight*. This is far more easily said than done. What do the results mean at the gut level? Does it all make sense? What are the implications? What further questions (and maybe simulations) are suggested by the results? Are you looking at the right set of performance measures?
- *Document what you've done*. You're not going to be around forever, so make it easier on the next person to understand what you've done and to carry things further. Documentation is also critical for getting management buy-in and implementation of the recommendations you've worked so hard to be able to make with precision and confidence.

By paying attention to these and similar issues, your shot at a successful simulation project will be greatly improved.

2.9 Exercises

2-1 For the hand simulation of the simple processing system, define another time-persistent statistic as the total number of parts in the system, including any parts in queue and in service. Augment Table 2-2 to track this as a new global variable, add new statistical accumulators to get its time average and maximum, and compute these values at the end.

2-2 In the preceding exercise, did you really need to add state variables and keep track of new accumulators to get the *time-average* number of parts in the system? If not, why not? How about the *maximum* number of parts in the system?

2-3 In the hand simulation of the simple processing system, suppose that the *queue discipline* were changed so that when the drill press becomes idle and finds parts waiting in queue, instead of taking the first one, it instead takes the one that [illegible] *shortest processing time* (this is sometimes called an *SPT* [illegible] work, you'll need to assign a second [illegible]