Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

CAPACITY AND PERFORMANCE ANALYSIS OF COMPUTER SYSTEMS

James N. Robinson

Capacity and Performance Management
Mead Data Central, a division of The Mead Corporation
P. O. Box 933
Dayton, Ohio, U.S.A., 45401

ABSTRACT

This paper presents a general tutorial for the use of discrete event simulation in support of software performance engineering (SPE) for computer systems focusing on client-server systems. Topics include definitions and tools of SPE, output measures of interest, a transaction approach to client-server computing simulation, the rationale for conducting SPE simulations versus alternatives, how and when simulation fits into a software development process, modeling techniques, input and output analysis issues, and verification and validation.

1 INTRODUCTION

The world's information systems are being transformed from central mainframe to distributed systems processing. The word "distributed" is defined here as processing accomplished by constellations of spatially separate and unique computers operating in parallel on a complex telecommunications web. Distributed processing is often related to client-server architectures supported by a variety of remote procedure call (RPC) mechanisms; however, systems can also be transaction based. Distributed processing can be analyzed in terms of work load and communications metrics produced by long or short lived programs in execution called "processes". Processes acting as clients or servers exist according to decisions reflecting spatial-temporal system management. Such distributed computing systems may reflect complex processing systems where the flow of data and work in terms of loads is a management challenge. Often, even small to moderately sized systems have interdependencies which cannot be captured effectively without very thorough dynamic analysis. Simulation of these systems is an effective tool for analysis.

Using simulation to analyze computer performance and capacity falls under software performance engineering (SPE). While a variety of tools are available, and valuable, for conducting SPE, simulation continues to stand out as a powerful tool which lends itself to a more complete understanding of loads and response times.

2 WHAT TO SIMULATE - SOFTWARE PERFORMANCE ENGINEERING (SPE)

Software performance engineering has been defined as a "method for constructing software systems to meet performance objectives" (Smith 1990). Smith's book, *Performance Engineering of Software Systems*, is an excellent place to start to understand the methodologies of SPE. She states: "The process begins early in the software life cycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance, before developers invest significant time in implementation. SPE continues through the detailed design, coding and testing stages to predict and manage performance of the evolving software, and to monitor and report actual performance against specifications and predictions. SPE methods cover performance data collection, quantitative analysis techniques, prediction strategies, management of uncertainties, data presentation and tracking, model verification and validation, critical success factors, and performance design principles."

SPE relates to the capacity and through-put a given computer system can sustain. Through-put may be described in terms of transactions per time unit, or in transaction response time. Capacity is described in elementary metrics such as percent CPU utilization, input/output (I/O) per time unit, memory utilization, storage utilization, and telecommunication capacity required in terms of bytes or bits passed per time unit over key channels. It is important to note here that the distinction between telecommunications modeling and computer SPE modeling is often blurred by the needs of analysis. For many models, it may be appropriate to treat the telecommunications as a "black box" which allows traffic at

specific rates. For other analyses, where comm links are critical components, a more focused telecommunications analysis may be required.

Required capacity is defined in simulation models by utilization of resources which are allocated to processes at discrete event times. The utilization of these resources can also be modeled analytically in queuing network models (QNM's). Capacity consumed in a model under a proposed operating scenario of forecast user transaction load can be used to scale the hardware system which must be provided to meet the needs of a proposed project. Either the proposed hardware platform(s) can be explicitly included as part of the model, or a generic platform set can be modeled. In the later case, it is necessary to compare the capacity required in terms of possible machines of choice through a standard frame of reference, for example, standard benchmarks, e.g. TPC-A's, TPC-C's, or SPECINTS. Benchmark scores are generally made available by various venders for their offered hardware. Publications such as the *SPEC Newsletter* produced by the Standard Performance Evaluation Corporation and *SIGMETRICS Performance Evaluation Review* are excellent sources of information on these benchmarks and the vender testing. If the capacity required can be scaled to a standard benchmark on the generic simulated machine, perhaps one designed to emulate an available test environment, that benchmark score can be used to acquire operational hardware in appropriate quantities.

Performance objectives are essential to conduct SPE analysis (Smith 1990). In practice, software development projects may not have specific design goals which reflect capacity and performance, particularly during early phases. For many commercial systems, any forecast of load and response time is linked to market and usage forecasts. Errors in these forecasts directly impact the validity of the output. Where the system being designed is evolutionary, for example an enhancement to an existing system where transaction rates and loads are well understood, usage forecasts which are captured in a model are better understood. In other cases where a totally new system is being brought in to serve a new operation, requirements may be much harder to produce. In either case, it is often necessary for the performance engineer to help form the performance and capacity requirements. Definition of the requirements is typically an evolutionary process which may require performance modeling, applied iteratively, to resolve.

The essence of a minimal client server system is shown in Figure 1 where major design elements (MDEs) under consideration in the design are the client on box A and the server on box B. This example might describe a simple text retrieval where the client represents a telecommunications and scheduling interface, and the server represents a data retrieval process. At a point in time (which may not be part of a model) the client and server process are started on the two machines. A user "transaction", a sequence of possible events defined by the transaction type, begins with a comm request received by A:

- Client processes task off wire using CPU-A, Comm I/O, I/O, and memory.
- Client application task using CPU-A, I/O and memory.
- Client constructs and sends RPC using CPU-A and ethernet comm I/O.
- RPC delay and bit count.
- Server comm I/O and process RPC using CPU-B.
- Server does required service "get text" using CPU-B, I/O and memory as required.
- Server marshals return RPC using CPU-B, and comm I/O.
- RPC delay, and bit count.
- Client process RPC using CPU-A, Comm I/O.
- Client does application processing using CPU-A, I/O and memory as required.
- Client puts message on the wire back to user CPU-A, comm I/O, and memory.

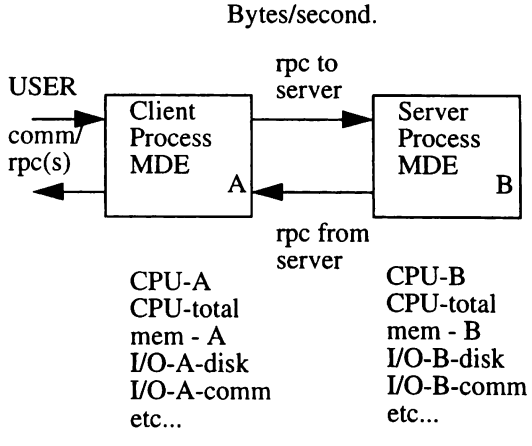

Figure 1: Elemental SPE Simulation Structure

This sequence can be readily captured in a discrete event simulation which models resource use and delays. The service performed by the server could involve a host of other servers, which have the server as client. A variety of events may be invoked which require dynamic memory allocation. Either of the processes may fork and execute child (or other) processes to perform specific duties. In turn, these child processes would require memory, CPU, disk, etc. to perform their tasks. The given

"transaction," as an entity, must queue for these resources, seize them, and free them in accord with the task timing imposed. Definition of multiple transaction scenarios represent the essence of computer system simulation modeling. In large transactional systems, these transactions can be represented by stochastic user states (Keezer, Fenic, and Nelson 1992). More complex system transaction based examples will be discussed during the tutorial.

3 WHEN TO SIMULATE

It is generally agreed that software and computer system development should follow a development process to be successful. A simulation analysis should also follow a process. These processes are linked through the conduct of SPE. A software development process model based on a so called "water fall" model has been defined for software development and is presented in Table 1 (Smith 1990). Simulation modeling processes have also been described (Pritsker 1985, Law and Kelton 1991, Banks and Carson 1984). The sequence of steps in a SPE simulation analysis closely follow the software development process and are also displayed in Table 1. Both of the processes are iterative and mutually supportive; however, this does not imply they are absolutely linked. Either of the processes can cycle faster and more often through iteration stages; however, it should be a goal for the simulation process to stay ahead of the development process providing guidance on efficiency whenever required.

In the early stages of modeling, simple, deterministic, mean value, transactional models may be sufficient. Such models have been called Execution Graph Models (EGMs) (Smith 1990). They may also involve queuing network analysis. These models allow the computation of CPU, memory, I/O and Comm on a per transaction basis. If the number of transactions in a time phase are known, the average resource utilization can be computed by summing resources used and dividing by time. Special languages for EGM's have been developed. Alternatively, these computations can be calculated on a spreadsheet, or captured using a simulation package without generation of random input or output. For some projects, this level of modeling may be sufficient, or the output of models at this level, a first modeling iteration, may help to define requirements and a preliminary design.

The next iteration phase is an assessment of the detailed design. The model here might be a full discrete event model of the system based on the now evolving design and initial coding done for prototypes. In this phase, developers are more likely to have some test data available from prototype design elements. Validity during this phase may still be low. The goal is a technical assessment of the capabilities and restrictions the design implies, and, most important, identification of major bottlenecks under stochastic load. Output of the models is used to suggest design changes and answer engineering questions on optimality of alternative design strategies. These questions often focus on architectural concepts such as load balancing among processes and machines, placement of function, and caching strategies etc. A formalized report on capacity and performance can be produced at this stage, and approximate optimization of resources to achieve performance goals can be presented based on various design configurations.

During the design, implementation, and integration phases, we again begin another cycle of the iteration of the simulation process. This phase is critical because actual observation of the operational behavior of the system is possible. Minor changes in the design may be required, but the emphasis should be on validating and improving model input and output. If all has been done well, the simulation process enters the final phases along with the project, i.e. maintenance and operation of the project and implementation of the simulation. It is our experience that phases are never completely clear cut. For major projects, there are often revisions of the system and revisions of simulations. Some simulation projects may never reach this final stage because it is not cost effective to maintain some models. On the other hand, where the system is large and maturing, for example a market penetration scenario where load increases dynamically, the simulation may be the only source of quick load forecasts to define needed incremental capacity as requirements change.

Table 1: Development Processes

S/W Devel. Process	Modeling Process
Requirements Analysis	Problem Definition
Functional Architecture	Data Collection
Preliminary Design	Input Data Analysis
Detailed Design	Model Definition
Coding	Model Coding
Unit Testing	Verification /Validation
Integration Testing	Output Analysis
Maint. & Operation	Implementation

4 WHY SIMULATE?

In many cases, queuing and queuing network models, as well as EGMs may be sufficient to address basic SPE issues. Simulation of computer software systems is expensive and time consuming. For large projects, modeling may take weeks or months to complete. Jain and others have emphasized this aspect (Jain 1991). There are many instances where this expense is justified and necessary. Gold reports, "Simulation is particularly important in a distributed environment, where a variety of factors - CPU, memory disk, networking resources, server capacity and network topology, for example - can affect overall performance and cause bottlenecks" (Gold 1993). This statement highlights a main reason to simulate, the ability to do analysis on the simultaneous dynamic utilization of resources. Generally, it is the complexity of large distributed systems which suggests a simulation approach. While a queuing network analysis, may be sufficient for baseline mean value analysis, the dynamic burstiness and state dependent shifts in load may require a more dynamic approach. Smith suggests that queuing network models may be insufficient to handle such entities as dynamically forked and branched processes, non-homogeneous demands, significant shifts in loads as jobs run through phases, and complex competition for resources requiring load balancing (Smith 1990). Often the non-homogeneous "waves" of load (caused by transaction bursts) of significant duration may impact distributed processes and cause slowed response. These waves may be difficult to capture. Mean value analysis may be insufficient to engineer the systems adequately to handle risk of extreme value loads. In any stochastic system of nodes, such as a distributed computing system which might be called a "virtual machine" (Heindel and Kasten, 1992), the ability of each node to handle the stochastic confluence of load in large surges must often be established. The dynamic visualization capabilities of simulation are uniquely capable in handling this task.

The complexity of building and running large SPE models has been a drawback in the past. In recent years, a variety of software has evolved capable of addressing large distributed computer system simulations. Such packages as GPSS/H, SLAMSYSTEM, ARENA, SIMSCRIPT II.5, SIMON, and WITNESS are adequate to the task (see Swain 1993 for others). The advent of graphical user interfaces (GUIs) on these packages has made them more capable in building large systems quickly.

5 HOW TO SIMULATE

Given the complexity of many SPE models, the goals for modeling must be crystal clear to prevent wasted effort. Usually, the goals of SPE simulation are a mixture of separate objectives. To achieve each goal may require different levels of detail in models. Ultimately, the goals support the needs of different customers such as developers, operators, and acquisition staff. The goals will also determine the amount of expertise required by a modeling team. A team is often required because one individual often will not possess all the skills necessary to understand the SPE aspects of large projects in allotted time. Actual modeling may be handled by one or two individuals; however, a supporting SPE group may provide additional expertise in current hardware and software advances, component modeling support, data acquisition, and final peer review.

Once the goals are determined, the detail level of analysis must be set. Many examples of highly detailed analysis abound in the literature. This high detail resides in the level of description of the low level hardware or software interfaces. Some example topics include: performance of cache in multi-processor systems (Zimmerman and Robinson 1993), single-bus versus two-bus multiprocessor comparison (Obaidata and Radaideh 1991), and performance analysis of a hypercube parallel processor architecture (Lamanna and Shaw 1991). Many of these analyses are hardware and operating system focused.

Drawbacks exist in performing highly detailed modeling for SPE. High detail models are costly and time consuming to develop and may require additional training for the modeler. They slow clock (wall) times to run if they are embedded in an SPE model of a large system. They may also require complex interfaces to higher level concurrent operations (McBeath and Keezer, 1993). Examples of high detail would be explicit modeling of the operating system with kernel(s), data base management systems, I/O processes etc. In some cases where benchmarks are not available, it may be necessary to model at this "basic principles" level, but it may often be found that too much detail slows both the modeling and the analysis without the benefit of greater pay-off. Great detail is often confused with validity or fidelity. In some cases of SPE, high detail is justified to support developers attempting to make optimal decisions on low level design. In other cases it may be "over modeling". It all depends on the goals. Good practice suggests high detail modeling should be done separately from overall models of large systems if possible, and the results of the low level models should be aggregated as parameter driven components in higher level models.

With goals and level of detail clearly in mind, the modeler incorporates a design into a simulation model. In the best of worlds, this design would be incorporated as, for example, the three models growing out of a detailed object-oriented modeling (OOM) and design process: object model, dynamic model, and functional model (Rumbaugh et al. 1991). The techniques of design are not

always widely known or followed as one might hope in development groups. Often, the modeler designs a model based on documentation which lays out a broad framework which may not clearly define dynamic activity and interfaces. In such cases, the modeler becomes more a contributor to the design by documenting a dynamic model for the project.

Documentation is rarely sufficient or timely to support simulation. There is no substitute for routine, recurring, one-on-one discussion with developers of the design. These discussions should ultimately result in a semi-formal conceptual model review of the system where the modeler demonstrates familiarity with the dynamic conceptual model. In this review the modeler presents assumptions, the dynamic concepts as interpreted, and the key elements of input known or required. This review states and clarifies the dynamic roles of major design elements (MDEs) which will be the primary system components modeled. The review presents key transactions, logically coupled sequences of dynamic MDE activity which absorb resources. The conceptual review is also critical as a restatement of the requirements in SPE relevant terms by establishing a scenario or set of scenarios which define the model. It clarifies the engineering requirements of the design, and may be a reinterpretation of external factors which must be explicitly accounted for, or it may open up the need for new ones (McBeath and Keezer 1993).

The actual modeling of the proposed design proceeds in parallel to the design process itself. Modeling must account for the resources which the software will consume (CPU, I/O, memory, and telecommunications bandwidth). The model must account for the environment in terms of supporting hardware and software architecture. This environment must be flexibly defined in the model to provide for anticipated "what if" analyses to be accommodated rapidly in the future. The environment may also be interpreted in terms of "logical domains" which may take many forms. A logical domain is represented by a pool of resources managed as a set, each having a set of constraints acting on them. For example, editorial staff may operate from UNIX workstations with common functionalities. This domain may draw on the resources of a fileserver which may be one of a set of fileservers with similar functionality. These fileservers may be served by a acquisition sequence involving data inflo through telecommunications gateways, dependent on automated and manual processing facilities. Here three domains may be designated in the model to capture data from the simulation which is relevant to managerial or engineering decisions. Behaviors (speeds, caching, availability of resources etc.) may vary with domain. The model must be built and instrumented to highlight each domain and its share of the whole. This may be implemented by using dummy domain resources which shadow other resources in the model.

The essence of software execution on a computer is the use of the CPU as a resource captured and freed by the transaction entities under the control of the operating system. In multiprocessor systems with a variety of constantly evolving operating systems, the choice of how much workload delay using CPU and memory on behalf of application processes is a difficult one. The first, and most difficult decision necessary, is how to correctly capture the operating system's impact on CPU consumption without modeling it explicitly. For example, in UNIX systems it is difficult to model both application user CPU time and system CPU time using the kernel (McBeath and Keezer 1993). Without explicitly modeling the operating system and kernel operations, modeling compromises must be made. An approach we have found successful, is to model CPU time as applications execute, and to model known CPU time normally spent on behalf of the system at the time of an input or output event or an RPC event. This allows visibility of key bottlenecks for CPU and I/O in the model. Since overhead, which is dependent on system CPU, is not accounted for explicitly in the model, a correction factor based on the assumed system overhead is applied to the output CPU utilization per domain to establish the number of multiprocessor machines required. For example, a 4 processor computer might be assumed to provide 1.7 effective processors under high I/O related load. Thus, the number of computers assumed required is the simulation utilization divided by 1.7.

CPU delay times on behalf of specific applications can be derived from benchmark tests or basic principles. One can assume, based on vender ratings, that the processor can handle "n" MIPS (million instructions per second), coupled with an assumption that a higher level language uses on average 20 machine instructions per high level instruction (Smith, 1993). Information on higher level instructions can be obtained from developers. Where possible, benchmark tests on comparable computers and processors are extremely helpful. Unfortunately, good benchmark runs are often difficult to get. Test machines and operating systems may not match performance of proposed machines for operations. It may be difficult to coordinate tests free of extraneous load.

At best, a model represents best engineering assumptions on application CPU timing. It is interesting that the state of the art is changing so rapidly in processing speeds, that whole data bases of past performance experiences are rendered questionable when migration is made to new systems; however, basic assumptions well stated on adapting this past knowledge to new systems are often adequate. This argues for a constant and energetic maintenance of a performance database for basic operations.

Smith (1993) describes memory as a "passive resource", that is one that is needed to do work, but does not actually do the work itself like CPU. While difficult to model effectively, an attempt to capture key elements of memory utilization is essential. For example, the short lived nature of processes, and the ability to execute and fork child processes dynamically indicates a need to track required memory at a gross level. Additionally, queuing for devices and processing may require large blocks of data to be maintained in memory at specific points in processing, such as at delivery across comm channels. A multitude of this type of queuing on a machine may cause thrashing for memory. In some systems greatly dependent on the availability of memory to support rapid response, particularly those involving large data bases, dynamic modeling of memory allocation may be necessary, much like CPU modeling. It is clear that memory modeling falls into the category of higher detailed modeling which is warranted in some circumstances, but may be tedious to implement effectively.

Like memory, storage, be it disk, tape or electronic storage, is another aspect of the model which may or may not be included depending on the needs of the analysis. Issues which constantly arise in a large system can involve the rate of accretion of data on storage devices, and the amounts which must be periodically backed up, stored permanently, or disposed of. Instrumentation data is such an item. A simulation model may track such needs routinely by maintaining state variables reflecting storage as the simulation progresses.

Input/output (I/O) load is a critical element in many systems. Delays are encountered during I/O which may relate to physical positioning of heads, determination of sector position, etc., and the CPU required to do the I/O itself. I/O costs may be intrinsically tied to the operating system involved, and the rate of the I/O is very dependent on the purpose of the software system. The selection of the hardware component suite to support the system greatly depends on the rates of I/Os. The simulation can be designed to capture both the CPU and the rates of I/O by using the CPU resource during I/O appropriately, and capturing rates of I/O using state variables. Often of interest, are the rates of I/O not only to storage devices, but to communication devices and networks. It often is of interest to capture the total extent of I/O in bytes. Likewise the rate of transfer of bytes may be captured using global state variables. Since the rate of change and amount of comm bandwidth in use are key elements in distributed systems design, the inclusion of the network at a low level, even as a black box is valuable.

6 INPUT DATA ANALYSIS

Input data analysis encompasses the techniques of choosing the parametric inputs to the model which are used to define delays and resource consumptions as the model runs. The definition of input for SPE simulations often requires considerable technical expertise. The delays in a discrete event model may be probabilistic or deterministic. If they are probabilistic, it is necessary to identify and fit underlying distributions to existing data, or to determine a heuristic approach to defining the distributions. Techniques for this input definition are amply covered in numerous texts, and are increasingly available in a variety of statistical software. See for example Law and Kelton (1991). In many instances, techniques presented are helpful; however, very large sample sizes can be collected from computer operations. This may be problematic as very large data sizes often predispose basic statistical tests of a distributional assumption to failure (rejection of the assumed distribution). Ultimately, the final judgement on a distribution to use may depend on careful reasoning.

One important aspect of SPE input analysis is the great breadth and depth of the data required and available, as well as the technology needed to obtain it. Often the data required requires as much knowledge of a complex system environment as the knowledge needed to model one. The services of a data specialist have been found helpful. Such an individual is able to employ techniques to gather and process the data using a variety of technologies. Often the capture of data requires observing, and possibly conducting benchmark studies; therefore, it is essential for a data specialist to be knowledgeable in a number of techniques and computer systems beyond statistics.

Recent advances in performance tools such as UNIX tools like Hewlett Packard's *Perf View*™, code structural analysis tools such as *Quantify*™, as well as standard utilities such as *vmstat, iostat*, and *perfmeter*, when coupled with dynamic instrumentation allow a detailed view of the observable capacity needs and performance of atomic (small time scale) component measurements of CPU, I/O and memory. In fact, the increasing amount of data, and widely varied format in which it can occur, has its own demands on analytical processing skills. Increasingly, facility with data base languages and SQL queries, text processing utilities like *perl*, and of course statistical or visualization packages which can handle large data structures such as *SAS*™, *SPSS*™ etc. are becoming invaluable for modeling support. It is increasingly important to have access to production and test platforms for these tools. These tools also become extremely important in validating simulation models.

7 OUTPUT ANALYSIS

Output analysis for SPE is like output analysis for other simulated systems. Differences occur in the length of time SPE simulations run, and in the categorization of the data produced. Excellent discussions are available, many with SPE related examples. See for example Welch (1983), Law and Kelton (1991), Jain (1991), and Banks and Carson (1984).

Aspects of SPE output analysis bear emphasis. One of these might be the emphasis on other than mean system behavior. One way to characterize the behavior of distributed processes is by their bursty behavior. Many of the processes analyzed exhibit very high variability. Some of this variability is due to the juxtaposition of stochastic processes with widely varying time scale periodicities. For example, during the operation of a system, we might observe constant activity with a mean frequency of 1 to 20 milliseconds between events, a middle ground of frequencies on the order of every 5 minutes or 300,000 ms, and even events on the order of every 15 minutes or 900,000 ms. Coupling these time scales with random fluctuations in flow which manifest themselves as low frequency shifts in load, makes the analysis of output problematic if only the "mean" is sought. Alternatives to a mean forecast are often required.

Focus on the goals of the analysis helps greatly. In some cases, operations personnel will be interested in the rate of occurrences of alarms where capacity utilization (performance) exceeds (goes below) nominal values, (e.g. 90% utilization, for a 5 minute period). In other cases the probability of exceeding a performance goal based on human factors considerations may be required. One is struck by the variety of statistical output considerations. In many cases, smoothing techniques such as exponential smoothing or moving average computations used to define patterns have been found helpful, particularly when compared across replicated runs and used in histograms or the analysis of empirical cumulative distribution functions. This approach focuses on the assessment of risk related to transients rather than mean response. One often finds the need to investigate these transient behaviors in apparently steady-state systems due to the difficulties induced by making long run times at small clock resolutions, typically the millisecond level. In many cases, it is impossible to achieve, or confirm, a so called steady state process due to the time resolution at which delays are measured. In some cases, we have found that runs must even be stopped due to the resolution limits of the simulation clock on personal computers.

Where steady state analysis is required, extraordinary measures may be needed to accelerate the advent of steady state using some form of intelligent initialization. We have often found injection of entities into a plausible starting population a valuable means to accelerate results. A variety of individualized approaches to dealing with high variability output may be necessary to capture relevant patterns in the underlying processes.

8 VERIFICATION AND VALIDATION

Verification of simulation models, answering the question of whether the model is built as intended, is extremely important, and that importance is well understood in the computer software development community. This means that conceptual reviews, developer interviews, code walk-throughs etc., typically are well understood and accepted. That is not to say they are never overlooked in the heat of pressing a product to completion. A determined program which lays out the ground rules of solid simulation development is critical to good SPE analysis.

Validation, the process of ensuring whether the simulation matches real world results, can be measured explicitly once software is implemented. The process of validation is also critical in a rapidly changing technological environment. It allows not only establishment of the validity of results, but also a basis for the next set of technologies or projects which in all probability will use some set of pieces from prior models. Data bases of capacity and performance connected with design elements in major evolving systems are a key to successful modeling programs. The evolutionary progress of major computer systems mandates a firm idea of where one has been as well as an eye to the next technology. A strong validation program supports this.

9 CONCLUSIONS

It is reasonable to use simulation as one of the primary tools with which to conduct SPE analysis of new and evolving computer systems. Simulation, particularly using modern GUI based model development environments, can provide an excellent basis to determine bottlenecks, do comparisons of competing designs, and even perform rough optimized capacity forecasts for new projects.

ACKNOWLEDGEMENTS

The author would like to thank the members of the CPM and development groups at MDC for all their kind help and insight into SPE simulation analysis. Special thanks go to Dave Withers for suggesting and reviewing this tutorial, and to Darby McBeath for her help in developing the ideas and reviewing the final paper.

REFERENCES

Banks, J., and Carson, J.S. 1984. *Discrete Event System Simulation.* Englewood Cliffs, NJ: Prentice Hall Gold, J. 1993. Simulation aims to predict software performance. *Software Magazine.* July: 15 - 19.

Heindel, L.E., Kasten, V.A. Workload Characterization and Analysis: Transition from Centralized Systems to Open Systems. *CMG 93 Proceedings*, ed. Bereznay, F. et al. 385 -394. Westmont, IL: Computer Measurement Group.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis. New York, NY: John Wiley and Sons.*

Keezer, W.S., Fenic, A.P., and Nelson, B.L. 1992. Representation of User Transaction Processing Behavior with a State Transition Matrix. *Proceedings of the 1992 Winter Simulation Conference*, Vol. 25, ed. Swain, J.J., Goldsman, D., Crain, R.D., Wilson, J.R. 1223-1231. Baltimore, MD: Association for Computing Machinery.

Law, A.M. and Kelton, W.D. 1991. *Simulation Modeling and Analysis.* New York, New York: McGraw Hill.

Lee, T. and Ghosh, S. 1994. A Distributed Approach to Real-Time Payments-Processing in a Partially-Connected Network of Banks: Modeling and Simulation. *Simulation.* Vol. 62:3 180 - 200.

McBeath, D.F. and Keezer, W.S. 1993. Simulation in Support of Software Development. *Proceedings of the 1993 Winter Simulation Conference*, Vol. 26, ed. Evans, G.W., Mollaghasemi, M., Russell, E.C., Biles, W.E. 1143-1151. Baltimore, MD: Association for Computing Machinery.

Obaidat, M.S. and Radaideh, M.A. 1991. A comparative simulation study of the performance of single-bus and two-bus multiprocessors. *Simulation.* Vol. 56: 19-17.

Rumbaugh, J., Blaha M., Premerlani, W., Eddy, F., and Lorensen, W. 1991. *Object Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall.

Lamanna, C.A. and Shaw, W.H. 1991. A performance study of the hypercube parallel processor architecture. *Simulation.* Vol. 56:3 185-196.

Smith, C. U. 1990. *Performance Engineering of Software Systems.* Reading, MA: Addison-Wesley.

Swain, J. 1993. Flexible Tools for Modeling. *OR/MS Today.* Vol. 20:6 62-78.

Szymanski, B.K. and Azzaro, S.H. *Proceedings of the 1990 Winter Simulation Conference*, Vol. 23, ed. Balsi, O., Sadowski, R.P., Nance, R.E. 831-838. Baltimore, MD: Association for Computing Machinery.

Vemuri, V. 1991. Simulation of a distributed processing system: A case study. *Simulation.* Vol. 56:5 302 - 315.

Welch, P. D. 1983. The Statistical Analysis of Simulation Results in *The Computer Performance Modeling Handbook.* New York, NY: Academic Press.

Zimmerman, S.L. and Robinson, J.P. Two Level Cache Performance for Multiprocessors. *Simulation.* Vol. 60:4 222 - 231.

AUTHOR BIOGRAPHY

JAMES N. ROBINSON is a software engineer in the Capacity and Performance Management group at Mead Data Central, home of LEXIS™ and NEXIS™, and an adjunct professor in the Engineering Management and Systems Department of the University of Dayton. Prior to retiring from the United States Air Force, he taught at the Air Force Institute of Technology, and served as the Director of the Graduate Space Operations program. His continuing research interests include visualization and statistical analysis of simulation input and output, time series, and other large scale data. Dr. Robinson has a Ph. D. in Operations Research from the University of Texas at Austin.