Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

SIMULATION IN SUPPORT OF SOFTWARE DEVELOPMENT

Darby F. McBeath
William S. Keezer

Mead Data Central, Inc.
P. O. Box 933
Dayton, Ohio 45401, U.S.A.

ABSTRACT

This paper discusses supporting the development of computer applications with simulation models from the early design phases through implementation. It presents a number of techniques we have found useful to communicate with customers and to build models of application systems. The greatest value in simulating developing systems is bringing all the information concerning a system to a single focus. Simulation support also greatly reduces the development time.

1 INTRODUCTION

For several years we have been supporting the design and development of software application systems with discrete event simulation. Jain (1991) considers the use of simulation to support performance analysis a difficult option, and lists a number of pitfalls and potential problems. The support of software development holds similar problems in our experience. However, we have been successful in our efforts to build fairly large and complex models in the same time frames as the systems under development.

Our motivation and goals are similar to those listed by Park and Getz (1992), though in a different environment. Our primary concerns are to analyze both system design and implementation with respect to resource requirements and performance measures. The models attain relatively large size over time through incremental growth, and, because the model schedule is parallel to the development, time constraints are often severe, particularly in the later stages of a project.

Though much has been written, for example, by Martin (1985), and by Sommerville (1992), on careful and accurate design, particularly with respect to logic flow and program module interaction, documented ways of looking at designs from a performance standpoint are few or non-existent. For example, McManus (1987) only considers performance of the most critical systems after the software is developed and is in testing. Many times intuition and experience are used to provide the desired judgement on design performance rather than quantitative techniques, and Card (1990) explicitly states that designs must be measured subjectively. Tanik (1991) uses Petri Nets to model system control for concurrency and interaction, but not for the system performance. However, by using simulation models of the designs, we can increase the accuracy of early estimates of performance and even provide early capacity estimates to support hardware procurement. In addition, we are able to anticipate problems and often indicate design changes before programming starts, thus saving effort at an early stage of development, rather than waiting until after the system is implemented and tested.

We also were able to use our models to make capacity requirement forecasts, estimate the behavior of the system from a user perspective, and help establish software and hardware configurations.

In this paper we discuss the environment in which both the computer applications and the models are built, the modeling process in general, the assumptions and simplifications used in modeling in detail, and some results and conclusions.

2 SOFTWARE DEVELOPMENT ENVIRONMENT

2.1 Software Development Environment

The software created by the developers provides the features and services of our on-line information service LEXIS® / NEXIS®. Thus both the functionality and the performance of the software are critical, and a large number of the departments in the company have an interest in or a need for the software being developed. The main customers of the model results are developers

and the production or information delivery system support groups. The developers are mainly interested in design feasibility early in the process and performance in the later stages. The production support people are interested in capacity issues and platform requirements from the beginning. Additionally, the groups that will use or market the product are as interested as the developers in the model results. Often they are not technically sophisticated, but must understand how the models work in general and the significance of the requirements, assumptions, and results, especially, as the modeling results may impact their planning for rollout of functions or addition of new features.

Many of the challenges we faced resulted from the environment and organization of work used by the software developers. Although the overall design is set early in a project, the actual implementation may vary significantly from the available documentation as problems are encountered and resolved. Especially during rapid development, the project documentation is scarce, and often out of date. The challenge then becomes one of keeping up with the changing designs and implementations.

Large projects are split among teams that develop functional units independently of each other. In addition, the membership and responsibilities often shift over time. This makes keeping track of the key sources of information an interesting problem at times. One solution was to develop good rapport with project leaders, who are generally the organizationally most stable participants.

Early in a project, the software developers may find the requests for design data bothersome. We found that positioning ourselves as having a mission of assisting them was very important. We were also fortunate in having strong management support for our work, both in the modeling area and in the overall software development area. This enabled us to easily obtain the desired information early in the process, before we had shown the utility of our work. Because we were sufficiently successful in our early efforts, all subsequent significant changes were brought to us for analysis.

The single greatest challenge was obtaining parameters for resource utilization in the model. Ideally such information would come from test results. However, we found that early testing of software is only for functionality, and that unless critical to the development effort or specifically required, internal instrumentation of the software modules was rarely created to measure resource usage or response times. At this stage, the judgement and experience of the modelers is important to providing reasonable estimates and place-holder values. Occasionally, there was a benchmark obtained on a module of the system, but these generally require analysis and recalibration to provide a good estimator for the model being developed. Once integration testing begins, performance information is much more readily available and can be incorporated to help calibrate the model. Since application development and preliminary testing is often carried out on platforms other than the final production platform, any parameter used in the model based on these results must be recalibrated to reflect the final implementation platform.

2.2 Model Development Environment

The models have a dual purpose; in addition to supporting development, they have to provide an accurate representation of the system for the production departments to use as performance analysis tools. Therefore, unlike Sussman in Crain, et al, (1992), we have to obtain predictive numerical results and answers, in addition to the insights and indicators of system behavior. This places a great importance on model validation, which, as just described, is often quite difficult until very late in the development cycle.

We are fortunate in the support we have available. Early in the development process, both the human factors group in the company and the modeling department analytical specialist were able to make estimates of the potential user behavior by extrapolating current user behavior (Keezer, Fenic, and Nelson, 1992). This allowed us to simulate realistic work load inputs from the start.

Since one of the goals of the modeling work is to create models that are independent of the simulated hardware platform, we had to have some means of creating parameters to describe the platform upon which the software is being implemented. Such parameters include the power of the CPU and the operating characteristics of the disk storage systems. Disk operating parameters are obtainable from the manufacturers in a reliable form. However, CPU power is subject to considerable interpretations by the manufacturers and by those producing benchmarks. We were very fortunate to have a specialist in benchmarking in the department, and he was able to provide us with measures of CPU power that could be used as ratios to a base value to adjust CPU costs in the models.

The members of the modeling team included one or more persons with little or no modeling experience. Their lack of experience was an asset, because it frequently forced us to clarify our own thinking in order to explain the work to them. Because everything was new to them, they frequently found problems that we had missed. From this experience, we recommend bringing in people new to modeling as members of the team. Co-op students or summer interns are one possible source.

Model development has been on SLAMSYSTEM® Release 3.0, from Pritsker Corp., on PC's (Pritsker,1986). On rare occasions model runs would be made on a mainframe when time was critical. For the size models we were running, we generally experienced a run time on the order of 12 to 24 hours. In an environment with severe time constraints, these kind of run times create a need for careful planning of work, and even then, when debugging problems occur, schedules may be impacted. Several times over the development of models, the compiled SLAMSYSTEM boundaries were reached and exceeded. Pritsker Corp., in all cases, was very responsive in sending us new compilations to allow the expanded models to run.

3 MODELING PROCESS

3.1 Overall Modeling Process

The modeling method used by our department is an iterative process. As described in (Withers,1992), the process steps for developing a model are the following:

1. Understand the customer needs.
2. Understand the system to be modeled.
3. Construct a conceptual model.
4. Review customer concept .
5. Develop a model.
6. Review model code .
7. Run model and analyze results.
8. Present results.

Communication with customers is most important in defining the goals of the analysis. "Every study must begin with a clear statement of the study's objectives; without such a statement there is little hope for success." (Law and Kelton, 1982) The statement of the objectives should not be done just at the beginning, but should be revisited throughout the project. The eight steps above are iterated many times, especially steps numbered 2-6 when changes and expansions of model detail are occurring. Once the concepts of the design of the software project are understood, the conceptual model can be constructed and reviewed by the customers. It is important at the concept review to have ALL the interested groups, especially the non-development groups, in the meeting. Our experience has shown that this is a major opportunity to clarify misunderstandings between groups and insure that everyone agrees on the overall design and goals of the software system. Once the model concepts are agreed upon, the model code can be developed and reviewed. The people most likely to use the model in the future and people knowledgeable of the language used to construct the model should participate in the code review. Next, the model can be run and the results analyzed. The final step is to share the results. In all results presentation, the model assumptions should be presented again, then the results will be understood and believed. All questions and comments were welcomed, and any issues were visibly recorded by an assigned scribe (a member of the modeling group).

Model validation at the conceptual phase is essential in building a model, and begins by communicating with the model customers. During information gathering, extracting information from the software designers and developers was difficult. Little design documentation existed, which caused us to rely on one-on-one discussions with the design experts.

One tool we have found to be very valuable for communication throughout the project is the use of flow diagrams. An example appears in Figure 1. The flow diagram follows high level logic of the model flow closely, but does not deal with the modeling details of reserving and using resources. We used a number of conventions in our flow diagrams. We always indicated the physical platforms involved in the flow, such as the user terminal, the new system platform, and other systems. Generally, we would indicate the overall physical configuration as a separate diagram (not shown), to provide an overall orientation before discussing the details of the transaction flows. Every data movement had a separate arrow between the source and the destination for that movement. This had the value of indicating how many times data had to flow between modules, terminals, or other systems to accomplish the purposes of the transaction. Any process that was initialized, used and terminated on a per transaction or per session basis was indicated by a rectangle. Permanent processes such as servers used by all transactions were indicated by circles. Hardware adaptors for communications or other details of the platform were not indicated. Disks were represented by one symbol for each functional use, such as spooling or data storage. If there were more than one spool in the system or more than one file type, there would be a symbol for each.

In this hypothetical example shown in Figure 1, transactions are generated at the user terminal, and pass to the platform via Path 1, where the communications processing may be simulated by modules A and B. Both would be software modules used to process the message at the central processor level. The transaction flow inside the platform is described in the following sequence:

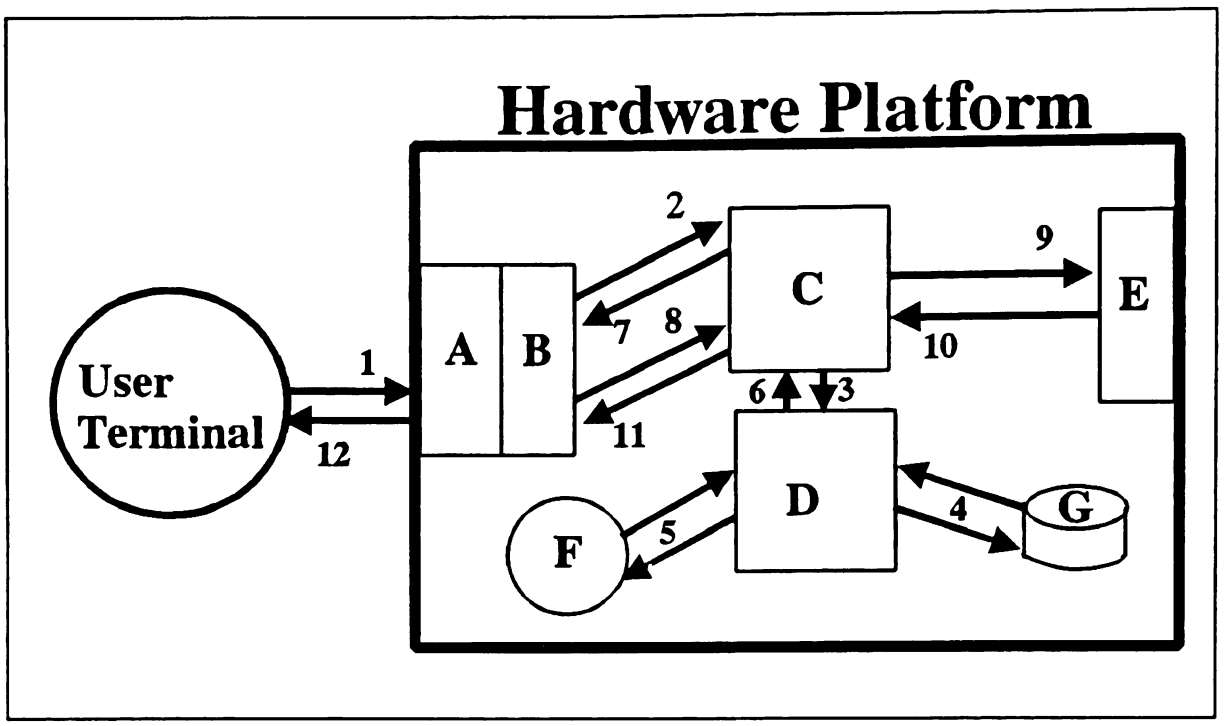

Figure 1. Example of a Flow Diagram. The large, unlabeled rectangle represents the hardware platform upon which the software system is to be implemented. The lettered boxes, A-F, represent software modules. The rectangular boxes are programs which are initialized, executed, and terminated. The circle, F, is a program which is available all the time. G represents a logical disk operation. The numbers 1-12 represent interaction and communication paths.

Path 2, request to module C from module B
Path 3, message to module D from module C
Path 4, Disk I/O by Module D
Path 5, request to Server F from module D
Path 6, return information to module C
Path 7, return information to module B
Path 8, request further information from module C
Path 9, request information from remote system
Path 10, return information to module C
Path 11, return information to module B.

Finally the transaction would return to the user terminal via Path 12. In the cases where there are more than one passage of information in one transaction, the path arrows are replicated to show this, such as (2,7) and (8,11).

Using flow diagrams proved to be an excellent way to describe the software applications and configuration of the system modeled. The customers, who have minimal modeling knowledge, can see the flow of the transactions in much the way they think of them, and in addition see interactions of which they may not have been aware and the overall picture of what is to be modeled. We have found that distributing these flow diagrams to the developers helps in obtaining more knowledge about what is actually being designed and developed, and the customers have a device for communicating change and additions to the design.

Cycling many times through this process from defining designs and requirements to delivering results has taught us much. Meeting with the developers of the software project early in the process forces them to pull together ideas in order to answer our questions. Meeting with all customers of the model often brings the entire project team together and closes some of the communication gaps in the designs.

For large development projects, where many complete cycles are necessary, we found that a cycle lasting six to eight weeks from concept to results provides the most benefit to the customers. If the cycle lasts more than eight weeks the customers may have forgotten many of the model assumptions and inputs as well as the details of the requested changes and additions. In addition, the software development cycle moves rapidly, and a model cycle of longer than eight weeks will not answer questions in a timely manner. Because of the validity checks and the necessity to conduct reviews, logic changes in the model generally required a minimum of six weeks to accomplish. We therefore created incremental releases requiring about six to eight weeks of work, prioritizing the desired changes. Following a

model release and results presentation, a series of "what-if" requests are frequently received. These include configuration changes, additions to the application system, or parameter changes.

"No good decisions are made from results and analysis that no one understands."(Garrambone,1992)

If the customers of models do not understand the inputs and assumptions going into the models, there is little hope they will understand or believe the analyses of the results. To assist the customers of the models in this understanding, we created a format for the results reviews that was never varied. We always presented the changed input assumptions and critical assumptions, whether changed or not, as the first section of the review. We then presented at a high level all the new logic introduced into the model. At both these steps all questions were invited and answered or recorded for later answer. Once we had an agreement that the inputs and changes were correct, we then presented the results and discussed them. This procedure is especially valuable when the results may not be the answer desired. By obtaining the buy-in up front as to the validity of the inputs and logic, the participants are less likely during the presentation of results to argue with the answer simply because they do not like it. In order to avoid giving the results before all assumptions were agreed on, we never handed out copies of the presentation foils ahead of time or at the meeting. We passed around a sign-up sheet from which we then sent out copies of the foils.

3.2 Level of Detail and Assumptions

A more detailed model is not always the better model. Developing a detailed model requires more development and debugging time. The more detailed the model the more data needs to be gathered for the model inputs. Furthermore, adding details and making assumptions about them when the data is unavailable or unknown could add unnecessary inaccuracies. As Sussman in (Crain, et all, 1992) points out, we must model the appropriate level of detail. Generalizing and making assumptions simplifies the model.

The sections of the model, where more analysis is required, should be expanded into more detail when needed by the customers. The approach we took from the beginning of the model development was to put generic modules as placeholders into the model where the details were unknown. These placeholders would use an estimated amount of the system resources. As the designs were created, the details could be easily incorporated into the placeholders' positions.

The following describes some of the areas where we decided to simplify models and make assumptions about the details.

3.2.1 Components outside the Application System Platform

Our policy was not to determine the resource consumption for those components outside the platform of interest. Such components as the work at the user's terminal or the work on a remote system were represented by fixed delays. If the user's terminal had software modules that interacted with the software on the hardware platform, estimated delays were incorporated for the processing times, but since the user terminal was assumed to be a single threaded, single user system, no resources or competition for resources were modeled. In the case of remote systems providing information, we generally had measures of the elapsed time to provide that information, and incorporated it as a delay in the processing.

3.2.2 UNIX - Operating System Simplifications

Some of the platforms under analysis ran the UNIX operating system. We decided not to distinguish between system CPU time and user CPU time in models, and therefore did not model the UNIX kernel. To do so would have required modeling details of kernel locking, priority scheduling, and time slicing. We would have had to estimate the proportion of user to system work for every module, thus doubling the number of CPU parameters in the model. System overhead, such as daemons and process accounting, was built into the CPU ratings we used, as was multiprocessing overhead in multiple CPU machines.

3.2.3 CPU Utilization

When projects begin, the software being modeled does not always exist, which forced us to make assumptions on the amount of CPU time each module would need. As the software was developed, we did not add any more detail, such as explicitly modeling subroutines. Our purpose was always to focus on the overall module behavior and the interaction between modules. Therefore, we refined our assumptions by running the code and extracting the measured CPU processing times for each software module from the system statistics reports. This approach kept the models simple and accurate for the intended purpose.

3.2.4 Process Scheduling

We had insufficient time and resources to model the multiprogramming and scheduling aspects of the UNIX operating system. We made the simplification that when an entity entered a simulated process, it would enter the queue for the CPU and stay in memory utilizing the CPU until it was done. This removes the time-slicing aspects of the operating system from the model. The impact of this would be on the distribution of the response times. The overall capacity requirement would be the same.

3.2.5 Disk I/O Operations

To simplify the models, we decided not to model disk I/O operations in detail. We used the manufacturer's average seek and latency times and values for the transfer time per character, as constants all at the spindle level. We assumed controllers were fast enough to never experience contention from the command and data transfers, and also assumed that controllers could have more than one outstanding operation. We also assumed that if a disk spindle were available it was the one to be used. This allowed us to avoid modeling data placement details.

3.2.6 Memory Allocation and Paging

Modeling the memory management process was too much detail for the scope of our models. We made the assumption that processes would not have to wait for paging, and we did not concern ourselves with memory allocation after the earliest releases of the model, in which we found that memory was not a constraint in our systems.

3.2.7 Data Base Management

Data base management systems, especially relational managers, are often considered to be major consumers of CPU resources and a source of response time increases. In our models, we created a fairly detailed sub-model of a relational manager, and simulated the behavior of non-relational managers as part of the simulation network. The relational manager sub-model was designed and built to be independent of the relational structures and operations for a given model, and to be easily modified to model relational managers other than the one of interest. This effort was well worth while, because by properly modeling the relational managers, we were able to place a proper perspective on their resource consumption and response time impacts. In one application system, though they were not trivial consumers of resources, they were not the resource hogs that folk lore often suggests they are.

3.3 Specific Techniques

Many techniques are developed and reused throughout each modeling project. This section describes a few of the techniques we have used.

3.3.1 Creating Hierarchical Sub-models in a Non-hierarchical System (SLAMSYSTEM Release 3.0)

The sub-modeling technique was created because we wished to keep the number of entity attributes as small as possible. We found reusing attributes in a submodel accomplished this need. The secondary goal was to create a reusable submodel that was easily incorporated into other system models. The first step was to copy the entity entering the submodel and its attributes. The copied entity would then enter the submodel, where its attribute space could be reused. The main entity would wait until its copy had finished execution in the submodel. Any information that the main entity would need at termination of the copy would next be obtained. Then the copy was terminated, leaving the main entity to continue throughout the rest of the model.

3.3.2 Modeling Communication Lines in an On-Line Transaction Processing (OLTP) System

We modeled individual sessions to ensure that we measured the utilization of the communication lines to the platform box, and that we guaranteed a simulated user was connected to the system through the same communication line resource throughout the session. To accomplish this, we created an array of the communication line indices with an indicator of whether they were free or in use, and by which entity. The entity, which represented a user session, was assigned the first available communication line. The index of this line was stored in an attribute and remained constant throughout the simulated session. At the termination of the entity, the communication line was marked available for reuse. This created a very accurate simulation of a communications pool.

We did not model communications protocols specifically. For asyncronous communications, we adjusted the effective data transfer rate to account for the start and stop bits and the inability of the software drivers to place data on the line at full bandwidth. For X.25 communications, we calculated the number of packets each transmission required, added 9 bytes of packet overhead for each packet, and then calculated delays based on a bandwidth utilization recommended by

the telecommunications engineers. For Bisynchronous communications, we adjusted the available bandwidth to account for polling, selecting, and acknowledgement overhead.

3.3.3 User Behavior Characterization

In an OLTP system each of the transactions modeled require different amounts of system resources, which means variations in user behavior can have a major impact on system resources. In the aggregate, users are fairly consistent, but individually have highly variable user behavior. Therefore, we needed a stochastic process to simulate user behavior in terms of transaction choices. The process we used, a transition matrix application, is found in Keezer, Fenic, and Nelson (1992).

3.3.4 Quickly Modeling Configuration Changes

The responsibility of the Systems Modeling group is to analyze systems on specified configurations. Based on the results there may be a choice to try a different configuration or even change the type of platform, thus changing the characteristics of the model's CPU processing times. By using a base CPU power rating and calculating the CPU processing times at the start of the simulation, the simulated platform could easily be changed. The calculation is based on the ratio of the CPU power values of the base and simulated platforms.

3.3.5 Print Delivery

Print despooling was modeled as two concurrent interacting processes. One process despooled data from a disk spool to a buffer area that was then delivered in smaller increments. The emptying process, which delivered the data to a printer, checked the large buffer contents, and issued a request for more data as necessary.

3.3.6 Initial Condition Bias and Analysis of Results

Because we were performing a steady state simulation, we were interested in removing the initial condition bias. We used the detection of initial-condition bias and data deletion techniques as described in Nelson (1991). We also used the analytical techniques as described in the same paper to analyze the results.

4 RESULTS

4.1 Impact to Projects

"The knowledge gained about a system while designing a simulation study may prove to be invaluable to understanding how the system really operates as opposed to how everyone thinks it operates."(Shannon,1992)

Most of the benefits of the modeling effort are realized in the first few steps during information gathering. As a result of the modeling efforts, numerous and extensive changes occurred in the development of the systems from the earliest stages. In several cases, the initial designs were shown to have severe capacity or performance impacts before application code was created. This caused a number of beneficial design changes at a much earlier than usual stage in development. In one outstanding case, the entire project was re-oriented as a result of the analysis of the capacity requirements of the initial design. We found that design flaws could surface after development had started, usually due to implementation choices or requirements not being compatible with the design or not achieving the expected capacity and performance goals.

Numerous bottlenecks in the flow between modules were identified and corrected, prior to testing.

As new features and functions were proposed, we were able to estimate their impact on capacity and performance and assist in their correct placement and implementation.

There were a number of capacity and configuration questions answered throughout the development process, both as to the amount of CPU power required and the amount of disk capacity. Many of these efforts were on a quick *ad hoc* basis. Quick turnaround was possible because of the modularity and high degree of parameterization of the models. Some efforts were made to estimate memory requirements, but in our systems this was the least constraining resource and the most difficult to model.

We were told by senior management that we had reduced the development time by a factor of 2 to 4.

4.2 Documentation

Throughout the development process, the presentation foils, the records of the configuration changes, the flow diagrams, our own documentation of all parameters used, and other model documentation provided a resource that continued to be referred to by the developers during testing and design changes. This became the single central source of all information on the systems being developed. This may be the single greatest contribution of a simulation project. It was especially valuable since we made an effort to keep it current with the actual rather than the formally documented design.

4.3 Further Uses

There are three uses for a system model after it has been developed. One, production analytical departments can use it to analyze performance questions both actual and hypothetical. Since our models are highly parameterized, questions of platform choice, function placement, system load, and configuration can be easily answered. Two, it can provide a theoretical performance target for the later stages of development when the system is under test. As a predictive tool at this stage, models are not as valuable; developers can sometimes change and test configurations faster than modelers can alter model logic. Third, it can provide a base of modeling code for assisting new projects, especially those that are offshoots of the current project.

5 CONCLUSIONS

Discrete event simulation:

- Provides insight to the performance of designs before any programs are developed.
- Gives hardware planners capacity and configuration information at a very early stage, allowing longer lead times and greater confidence in hardware procurement.
- Describes the system performance from the users perspective, showing where there are problems in transactions or communications.
- Shortens the development time frame with early detection of problems and decreases the effort on code that must be reworked due to design problems.

Important ideas to remember:

- The modeling process must be approached incrementally from the start, with cycles of 6-8 weeks to develop new, changed, or more detailed simulations of the software system.
- Care must be taken to communicate frequently and accurately with the developers.
- It is essential that documentation be properly maintained.

Discrete event simulation has a great value in providing support for large software development projects, and perhaps its greatest value lies in the integration and documentation of all the designs and subdesigns in one place and revealing their various, sometimes unexpected, interactions.

ACKNOWLEDGMENTS

We wish to acknowledge the many fruitful discussions with the members of the Systems Modeling Department concerning the models during development. Our thanks also to Jim Robinson, Jack Eddington, Neal Keller, Linda Showalter, and Dave Withers for reviewing this paper during preparation. Their comments were helpful and very much to the point.

REFERENCES

Card, D.N. and Glass, R.L. 1990. *Measuring Software Design Quality*. Englewood Cliffs, New Jersey: Prentice Hall.

Crain, R.C., et al 1992. Twenty-Fifth Anniversary Keynote Address. *Proceedings of the 1992 Winter Simulation Conference*, Vol. 25, ed. Swain, J.J., Goldsman, D., Crain, R.D., Wilson, J.R. 3-33. Baltimore, MD: Association for Computing Machinery.

Garrambone, M.W. 1992. An Overview of Airland Combat Modeling and Simulation. *Proceedings of the 1992 Winter Simulation Conference*, Vol. 25, ed. Swain, J.J., Goldsman, D., Crain, R.D., Wilson, J.R. 1130-1138. Baltimore, MD: Association for Computing Machinery.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis*. New York: 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. 1982. *Simulation Modeling and Analysis*. New York: McGraw-Hill Book Co.

Martin, J.R. 1985. *System Design from Provably Correct Constructs*. New York: Prentice Hall+.

McManus, J.I. 1987. "How Does Software Quality Assurance Fit In?" Chapter 2 in *Handbook of Software Quality Assurance*. New York: Van Nostrand Reinhold.

Nelson, B.L. 1992. Statistical Analysis of Simulation Results. Chapter 102 in *Handbook of Industrial Engineering*, 2nd ed. New York: John Wiley and Sons.

Park, C.A., and Getz, T. 1992. The Approach to Designing a Future Pharmaceutical Manufacturing Facility. *Proceedings of the 1992 Winter Simulation Conference*, Vol. 25, ed. Swain, J.J., Goldsman, D., Crain, R.D., Wilson, J.R. 933-935. Baltimore, MD: Association for Computing Machinery.

Pritsker, A.A.B. 1986. *Introduction to Simulation and SLAM II* . New York: John Wiley and Sons.

Shannon, R.E. 1992. Introduction to Simulation. *Proceedings of the 1992 Winter Simulation Conference*, Vol. 25, ed. Swain, J.J., Goldsman, D., Crain, R.D.,

Wilson, J.R. , 65-73. Baltimore, MD: Association for Computing Machinery.

Sommerville, I. 1992. *Software Engineering*, 4th Ed. New York: Addison-Wesley.

Tanik, M.M. and Chan, E.S. 1991. *Fundamentals of Computing for Software Engineers*. New York: Van Nostrand Reinhold.

Withers, D.H. 1992. Protocols for Discrete Event Simulation. Presented at the 1992 Joint National ORSA/TIMS Meeting, Orlando, Florida.

AUTHOR BIOGRAPHIES

DARBY F. MCBEATH has been a Software Engineer with Mead Data Central (MDC) for over three years. She is currently with the Systems Modeling Department where she develops application system models. Her work also includes capacity and performance analyses on developing systems. Previously at MDC she has worked in the human factors area. She holds a BS degree in Mathematical Sciences from the University of North Carolina at Chapel Hill and is a member of the ACM.

WILLIAM S. KEEZER has been with Mead Data Central (MDC) for over five years and is currently a Staff Analyst focusing on system and communications performance issues. Before coming to MDC, he was an in-house consultant on OLTP system performance problems for the Data Pathing Division of NCR. He holds B.S. and Ph.D. degrees from the University of Oklahoma, and is a member of the ACM.