Chapter 14

Proposed Systems

14.1. Introduction

The preceding two chapters have discussed the parameterization of queueing network models of existing systems and evolving systems. In this chapter we consider models of proposed systems: major new systems and subsystems that are undergoing design and implementation.

The process of design and implementation involves continual tradeoffs between cost and performance. Quantifying the performance implications of various alternatives is central to this process. It also is extremely challenging. In the case of existing systems, measurement data is available. In the case of evolving systems, contemplated modifications often are straightforward (e.g., a new CPU within a product line), and limited experimentation may be possible in validating a baseline model. In the case of proposed systems, these advantages do not exist. For this reason, it is tempting to rely on seat-of-the-pants performance projections, which all too often prove to be significantly in error. The consequences can be serious, for performance, like reliability, is best designed in, rather than added on.

Recently, progress has been made in evolving a general framework for projecting the performance of proposed systems. There has been a confluence of ideas from software engineering and performance evaluation, with queueing network models playing a central role. The purpose of this chapter is to present the elements of this framework. In Section 14.2 we review some early efforts. In Section 14.3 we discuss, in a general setting, some of the components necessary to achieve a good understanding of the performance of a proposed system. In Section 14.4 we describe two specific approaches.

14.2. Background

User satisfaction with a new application system depends to a significant extent on the system's ability to deliver performance that is acceptable and consistent. In this section we describe several early attempts at assessing the performance of large systems during the design stage. Some common themes will be evident; these will be discussed in the next section.

In the mid 1960s, GECOS III was being designed by General Electric as an integrated batch and timesharing system. After the initial design was complete, two activities began in parallel: one team began the implementation, while another developed a simulation model to project the effects of subsequent design and implementation decisions.

The simulation modelling team came out second best. The model was not debugged until several months after a skeletal version of the actual system was operational. Thus, many of the design questions that might have been answered by the model were answered instead by the system. The model could not be kept current. The projections of the model were not trusted, because the system designers lacked confidence in the simulation methodology.

This attempt to understand the interactions among design decisions throughout the project lifetime failed. Other attempts have been more successful.

In the late 1960s, TSO was being developed as a timesharing subsystem for IBM's batch-oriented MVT operating system. During final design and initial implementation of the final system, an earlier prototype was measured in a test environment, and a queueing network model was parameterized from these measurements and from detailed specifications of the final design.

The average response time projected by the model was significantly lower than that measured for prototype. However, the design team had confidence in the model because a similar one had been used successfully for MIT's CTSS system (see Section 6.3.1). The team checked the prototype for conformance with specifications and detected a discrepancy: the scheduler had been implemented with an unnecessary locking mechanism that created a software bottleneck. When this was corrected, the projections of the model and the behavior of the prototype were compatible.

In the early 1970s, MVS was being designed and developed as a batch-oriented operating system for IBM's new family of virtual memory machines. A simulation model was developed for an early version of this system, OS/VS2 Release 2. The model's purpose was to provide performance information for system designers.

Model validation was a problem. In the design stage, key model parameters were represented only as ranges of values. Performance projections were checked for reasonableness, to ensure that the model represented the functional flow of work through the system. This type of sensitivity analysis compensated for the lack of precise parameter values. The system was changing constantly during design and implementation. To reduce this problem, the model builders maintained a close working relationship with the system designers and implementors.

This modelling effort was considered to be a success, because several of its recommendations had direct, beneficial effects on the design of the system.

In the mid 1970s, the Advanced Logistics System (ALS) was under development for the U.S. Air Force. After the design was completed, during initial implementation, a modelling study was undertaken to determine the bottlenecks in the design and to recommend alternate designs yielding better performance. Hierarchical modelling, as described in Chapter 8, was applied. Four major subsystems were identified in ALS: CPU and memory, system disks, database disks, and tapes. A hierarchical model was structured along these lines, dividing the modelling task into manageable components. Parameter values came from a combination of measurements and detailed specifications.

Both analytic and simulation solutions of the model were obtained. Most ALS features could be captured in the analytic solution. Simulation was used to validate the analytic results and to explore certain system characteristics in more detail.

The modelling study predicted that as the workload increased, the first bottleneck would be encountered in the system disk subsystem, and the next in the CPU and memory subsystem. Both predictions were verified in early production operation. Thus, the study was judged a success.

Each successful project that we have described used a different underlying approach: an analytic model for TSO, a simulation model for MVS, and hierarchical analytic and simulation models for ALS. However, these projects shared a number of underlying principles. In the next section, we include these and other principles in a general framework for studying the performance of proposed systems.

14.3. A General Framework

Unfortunately, it is not common to attempt to quantify the performance of proposed systems. There are two major reasons for this:

- Manpower devoted to performance projection is viewed as manpower that otherwise could be devoted to writing code and delivering the system on time.
- There is no widely accepted approach to integrating performance projections with a system design project.

The first of these points is rendered invalid by the false sense of economy on which it is based: the implications of misguided design decisions for the ultimate cost of a system can be enormous. The second of these points is becoming less significant as aspects of a general framework begin to emerge. These are the subject of the present section.

14.3.1. The Approach

Performance is not the domain of a single group. Thus, performance projection is best done in a team environment, with representation from groups such as intended users, software designers, software implementors, configuration planners, and performance analysts. By analogy to software engineering, the team would begin its task by conducting a *performance walkthrough* of a proposed design. A typical walkthrough would consist of the following steps:

- The intended users would describe anticipated patterns of use of the system. In queueing network modelling terms, they would identify the workload components, and the workload intensities of the various components.
- The software designers would identify, for a selected subset of the workload components, the path through the software modules of the system that would be followed in processing each component: which modules would be invoked, and how frequently.
- The software implementors would specify the resource requirements for each module in system-independent terms: software path lengths, I/O volume, etc.
- The configuration planners would translate these system-independent resource requirements into configuration-dependent terms.
- The performance analysts would synthesize the results of this process, constructing a queueing network model of the system.

Various parts of this process would be repeated as the performance analysts seek additional information, as the design evolves, and as the results of the analysis indicate specific areas of concern. An important aspect of any tool embodying this approach is the support that it provides for this sort of iteration and successive refinement.

It should be clear that what has been outlined is a methodical approach to obtaining queueing network model inputs, an approach that could be of value in any modelling study, not just an evaluation of a proposed system. (For example, see the case study in Section 2.4.)

It also should be clear that this approach, since it forces meaningful communication between various "interested parties", can be a valuable aid in software project management.

14.3.2. An Example

Here is a simple example that illustrates the application of this general approach. A store-and-forward computer communication network is being designed. Our objective is to project the performance of this network, given information about the planned usage, the software design, and the supporting hardware.

The topology (star) and the protocol (polling) of the network are known. The system is to support three kinds of messages: STORE, FORWARD, and FLASH. From the functional specifications, the arrival rate, priority, and response time requirement of each message type can be obtained. Each message type has different characteristics and represents a non-trivial portion of the workload, so it is natural to view each as a separate workload component and to assign each to a different class. Given knowledge of the intended protocol, a fourth class is formulated, representing polling overhead. Further refinements of this class structure are possible during project evolution.

The software specifications for each class are imprecise in the initial stages. Only high-level information about software functionality, flow of control, and processing requirements are available. A gross estimate of CPU and I/O resource requirements for each class is obtained. The CPU requirement specifies an estimated number of instructions for each message of the type, and an estimated number of logical I/O operations. For STORE messages, as an example, the I/O consists of a read to an index to locate the message storage area, a write to store the message, and a write to update the index. No indication is given here about file placements or device characteristics. Instead, the logical properties of the software are emphasized, to serve as a basis for further refinement when the software design becomes more mature.

Physical device characteristics are identified: speed, capacity, file placement, etc. A CPU is characterized by its MIPS rate and its number of processors. A disk is characterized by its capacity, average seek time, rotation time, transfer rate, and the assignment of files to it. From consideration of the software specifications and the device characteristics, service demands can be estimated. As a simple example, a software designer may estimate 60,000 CPU instructions for a STORE message, and a hardware configuration analyst may estimate a CPU MIPS rate of .40. This leads to a STORE service demand for the CPU of .15 seconds. This admittedly is a crude estimate, but it serves as a basis, and more detail can be incorporated subsequently.

At this point, a queueing network model of the design, incorporating classes, devices, and service demands, can be constructed and evaluated to give an initial assessment of performance. Alternatives can be evaluated to determine their effect on performance. Sensitivity analyses can be used to identify potential trouble spots, even at this early stage of the project.

One of the strengths of this approach is the ability to handle easily changes in the workload, software, and hardware. In the example, no internal module flow of control was specified and processing requirements were gross approximations. As the design progresses, the individual modules begin to acquire a finer structure, as reflected in Figure 14.1. This can be reflected by modifying the software specifications. This structure acquires multiple levels of detail as the design matures. The submodules at the leaves of the tree represent detailed information about a particular operation; the software designer has more confidence in the resource estimates specified for these types of modules. The total resource requirements for a workload are found by appropriately summing the resource requirements at the various levels in the detailed module structure. Software specifications thus can be updated as more information becomes available.

The important features we have illustrated in this example include the identification of workload, software, and hardware at the appropriate level of detail, the transformation of these high-level components into queueing network model parameters, and the ability to represent changes in the basic components.

14.3.3. Other Considerations

The design stage of a proposed system has received most of our attention. This is where the greatest leverage exists to change plans. However, it is important to continue the performance projection effort during the life of the project. Implementation, testing, and

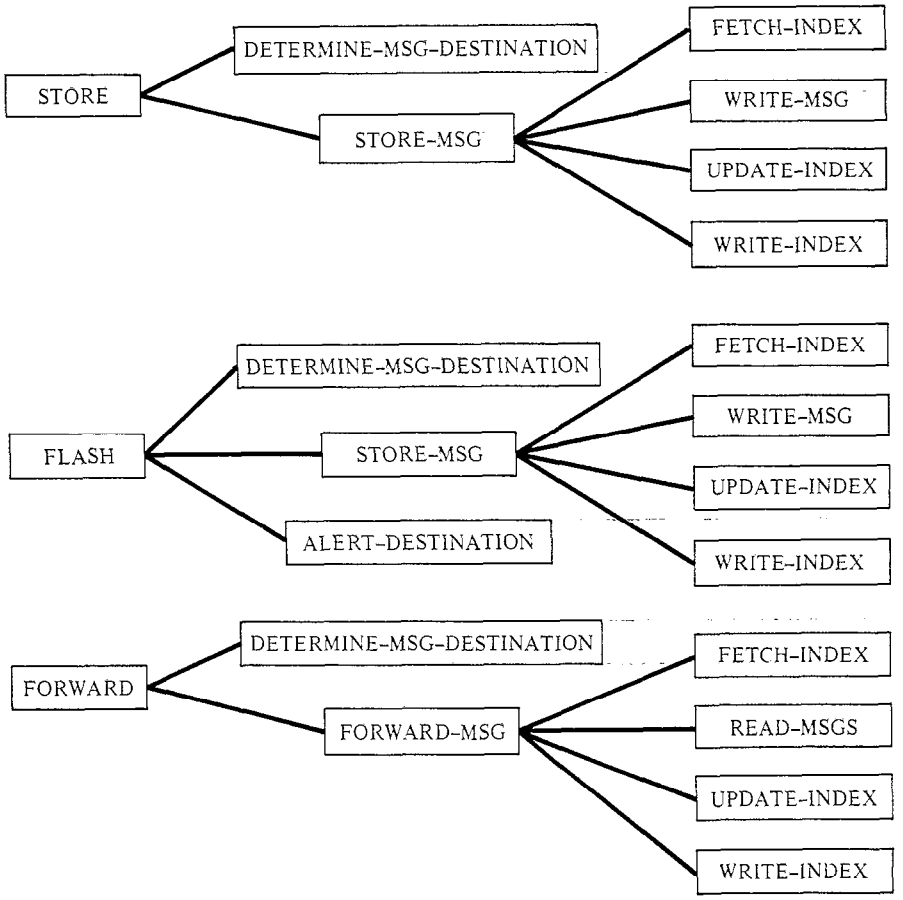


Figure 14.1 – Refinement of Software Specifications

maintenance/evolution follow design. Estimates indicate that the largest proportion of the cost of software comes from the maintenance/evolution stage.

Given the desirability of tracking performance over the software lifetime, it is useful to maintain a repository of current information about important aspects of the project (e.g., procedure structure within software modules). If the repository is automated in database form, software designers and implementors are more likely to keep it current.

A prerequisite for the success of the approach we have outlined is that management be prepared to listen to the recommendations rather than adopting an expedient approach. Budgeting time and manpower for performance projection may lengthen the development schedule somewhat, but the benefits can be significant.

A final important factor is the ability to turn this general framework into specific working strategies. In the next section, we describe two recent tools that are examples of attempts to do so.

14.4. Tools and Techniques

14.4.1. CRYSTAL

CRYSTAL is a software package developed in the late 1970s to facilitate the performance modelling of proposed and evolving application systems.

A CRYSTAL user describes a system in three components: the *module specification*, the *workload specification*, and the *configuration specification*. These specifications are inter-related, and are developed in parallel. They are stated in a high-level *system description language*.

- The *module specification* describes the CPU and I/O requirements of each software module of the system in machine-independent terms: path lengths for the CPU, and operation counts to various files for I/O.
- The *workload specification* identifies the various components of the workload, and, for each component, gives its type (i.e., transaction, batch, or terminal), its workload intensity, and the modules that it uses.
- The *configuration specification* states the characteristics of hardware devices and of files.

From these specifications, CRYSTAL calculates queueing network model inputs. These are supplied to an internal queueing network evaluation algorithm, which calculates performance measures.

We illustrate some of the important aspects of CRYSTAL by describing its use in modelling a proposed application software system. An insurance company is replacing its claims processing system. CRYSTAL is used to determine the most cost-effective equipment configuration to support the application.

As a first step, the workload components are identified in the workload specification. Many functions are planned for the proposed system, but the analyst determines that five will account for more than 80% of the transactions. These include, for example, Claims Registration. (This information comes from administrative records.)

Since the planning of this system is in its preliminary stages, it is not possible to say with certainty how the system will be structured into modules. The analyst decides initially to define one module

corresponding to each of the five workload components. This information is represented in both the workload and the module specification. (Naturally, this is an area where the appropriate level of detail will vary as knowledge of the system evolves.)

For each module, resource requirements are stated in the module specification. The units of CPU usage are instructions executed. There are two components: application path length and support system path length. In the case of the example, benchmarks of similar modules currently in use provide information for application path length. Where no benchmark exists, the logical flow of the software is used to provide estimates. For support system path length, major system routines are examined in detail to provide estimates; some benchmarks also are done. The units of I/O usage are number of physical I/O operations. The analyst determines these, beginning from a logical view of each module, and taking into account the file structure to be used.

The major application files and their sizes are part of the configuration specification. (These files correspond to those referred to in the I/O component of the module specification.) Initially, a simple file structure is proposed, but eventually file indices and database software will be introduced. In addition, a series of entries describe the devices of the system, e.g., for a disk, its transfer rate, seek time, rotation time, and a list of its files.

When the system description is complete, CRYSTAL can calculate queueing network model inputs and obtain performance measures. For example, response times can be projected for the baseline transaction volume and hardware configuration. If the results are satisfactory when compared to the response time requirement stipulated for the application, projections can be obtained for increased transaction volume by adjusting the arrival rates of the relevant workloads in the workload specification. Hardware alternatives can be investigated in a similar manner.

This concludes our description of CRYSTAL. The major activities in using this tool are completing the module specification, the workload specification, and the configuration specification. The study described here occurred during the initial stages of a project. As noted before, additional benefits would arise if the study were extended through the lifetime of the project. Better resource estimates would be available from module implementation, and the ability of the configuration to meet the response time requirement could be re-evaluated periodically.

14.4.2. ADEPT

The second technique to be discussed is ADEPT (A Design-based Evaluation and Prediction Technique), developed in the late 1970s.

Using ADEPT, resource requirements are specified both as average values and as maximum (upper bound) values. The project design is likely to be suitable if the performance specifications are satisfied for the upper bounds. Sensitivity analyses can show the system components for which more accurate resource requirements must be specified. These components should be implemented first, to provide early feedback and allow more accurate forecasts.

The software structure of the proposed application is determined through performance walkthroughs and is described using a graph representation, with software components represented as nodes, and links between these components represented as arcs. Because the software design usually results from a top-down successive refinement process, these graphs are tree-structured, with greater detail towards the leaves. An example is found in Figure 14.2, where three design levels are shown. Each component that is not further decomposed has a CPU time estimate and a number of I/O accesses associated with it.

The graphs are analyzed to determine elapsed time and resource requirements for the entire design by a bottom-up procedure. The time and resource requirements of the leaf nodes are used to calculate the requirements of the nodes one level up, and so on up to the root node. A static analysis, assuming no interference between modules, is performed to derive best case, average case, and worst case behavior. The visual nature of the execution graphs can help to point out design optimizations, such as moving invariant components out of loops.

Additional techniques handle other software and hardware characteristics introduced as the design matures. These characteristics include data dependencies (for which counting parameters are introduced), competition for resources (for which queueing network analysis software is used), and concurrent processing (in which locking and synchronization are important).

ADEPT was used to project the performance of a database component of a proposed CAD/CAM system. Only preliminary design specifications were available, including a high-level description of the major functional modules. A small example from that study will be discussed. A transaction builds a list of record occurrences that satisfy given qualifications, and returns the first qualified occurrences to the user at a terminal. It

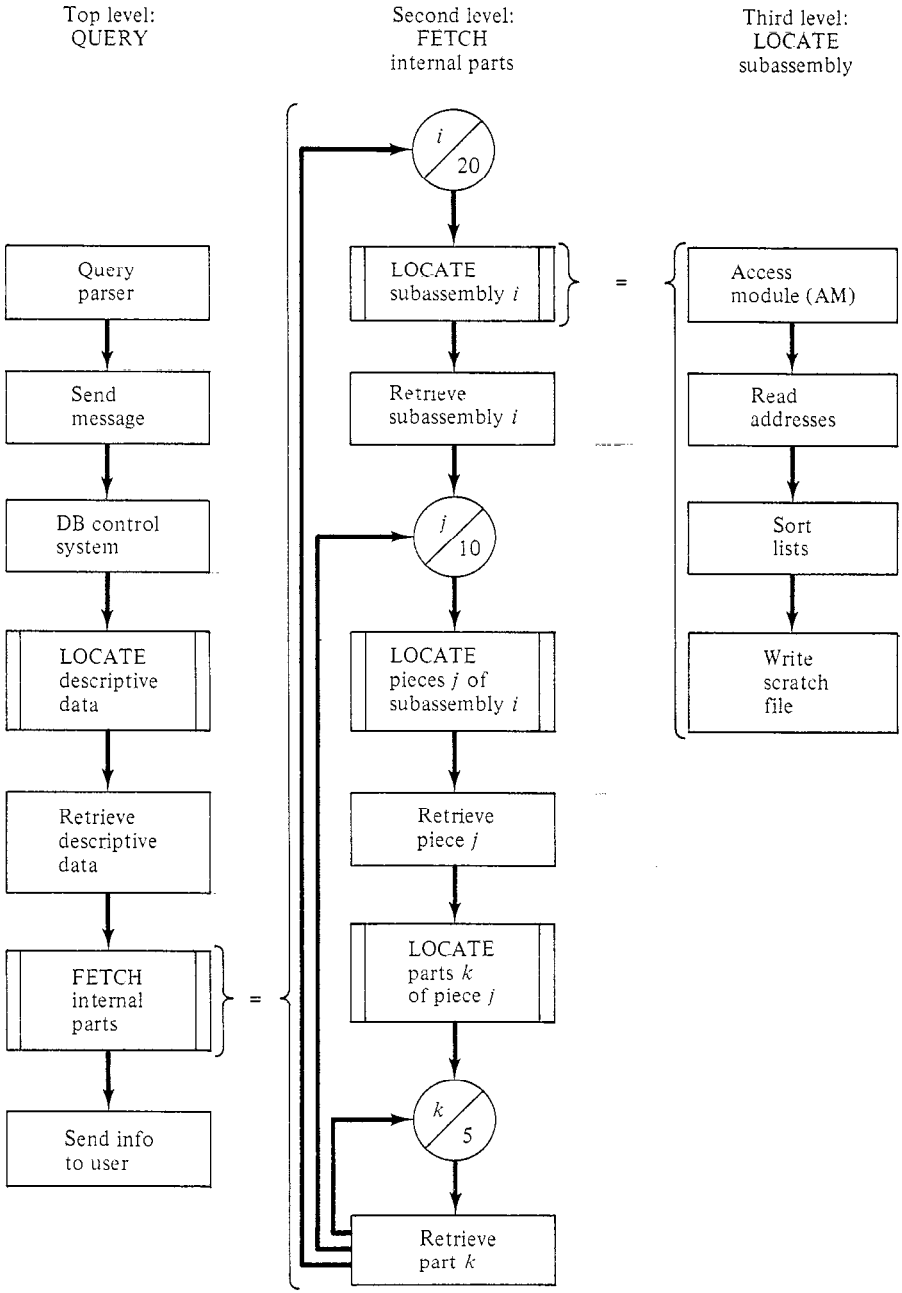
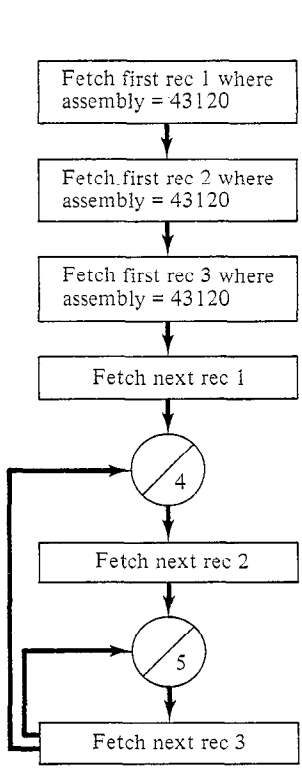


Figure 14.2 – Example Execution Graphs

issues FIND FIRST commands to qualify record occurrences and FIND NEXT commands to return the occurrences. The execution graphs for the FIND commands have the structure shown in Figure 14.3.

The performance goal for processing this transaction was an average response time of under 5 seconds, when the computing environment was a Cyber 170 computer running the NOS operating system. A performance walkthrough produced a typical usage scenario from an engineering user and descriptions of the processing steps for the FIND commands from a software designer. Resource estimates for the transaction components were based on the walkthrough information. Many optimistic assumptions were made, but the best case response time was predicted to be 6.1 seconds, not meeting the goal (see Figure 14.3). About 43% of this elapsed time (2.6 seconds) was actual CPU requirement. Thus, it was clear at the design stage that response times would be unacceptably long because of excessive CPU requirements.



		CPU (secs)	I/Os	Elapsed (secs)
Fetch first rec 1 where assembly = 43120		0.488	27	1.514
Fetch first rec 2 where assembly = 43120		0.488	27	1.514
Fetch first rec 3 where assembly = 43120		0.488	27	1.514
Fetch next rec 1		0.116	1	0.154
4				
Fetch next rec 2	Average	0.116	1	0.154
	Total	0.464	4	0.616
5				
Fetch next rec 3	Average	0.116	1	0.154
	Total	0.580	5	0.770
		2.624	91	6.082

Figure 14.3 – Transaction Steps and Projections

The application system has been implemented. Although actual parameter values were different in the running system than in the design, CPU bottlenecking still was present, more than a year after it was predicted. This demonstrates the success of the ADEPT approach. (The specific corrective advice provided by the performance analysts using ADEPT was not acted on, because it would have caused slippage in the delivery schedule for the system. However, the performance problems that arose resulted in delay and dissatisfaction anyway.)

This study shows that it is possible to predict with reasonable accuracy resource usage patterns and system performance of a large software system in the early design stage, before code is written. It also is possible to achieve these benefits without incurring significant personnel costs. This example was part of a project staffed by a half-time performance analyst and took approximately one person-month of work.

14.5. Summary

The process of design and implementation involves continual tradeoffs between cost and performance. Quantifying the performance implications of various alternatives is central to this process. Because doing so is challenging and requires resources, it is tempting to rely on seat-of-the-pants performance projections. The consequences of doing so can be serious, for user satisfaction with a system depends to a significant extent on the system's ability to deliver acceptable performance.

We began this chapter with a description of several early experiences in projecting the performance of proposed systems. We then discussed various aspects of a general approach to the problem. Finally, we studied two recent attempts to devise and support this general approach. We noted that projecting the performance of proposed systems requires a methodical approach to obtaining queueing network model inputs, an approach that could be of value in any modelling study. We also noted that the process of performance projection can be a valuable project management aid, because it serves to structure and focus communication among various members of the project team.

14.6. References

The early attempts at projecting the performance of proposed systems, discussed in Section 14.2, were directed not towards devising general approaches, but rather towards addressing the particular problems of specific systems. The study of GECOS III was described by Campbell and

Heffner [1968]. The study of TSO was described by Lassettre and Scherr [1972]. The study of OS/VS2 Release 2 was described by Beretvas [1974]. The study of ALS was described by Browne et al. [1975]. A good summary of these attempts appears in [Weleschuk 1981].

We have described two recent attempts at devising and supporting general approaches. CRYSTAL was developed by BGS Systems, Inc. [BGS 1982a, 1982b, 1983]. The examples in Sections 14.3.2 and 14.4.1 come from internal BGS Systems reports, as does Figure 14.1. ADEPT was developed by Connie U. Smith and J.C. Browne. The case study in 14.4.2 was conducted by Smith and Browne [1982]; Figure 14.3 comes from this paper. Other good sources on ADEPT include [Smith 1981] (the source of Figure 14.2), and [Smith & Browne 1983].

[Beretvas 1974]
T. Beretvas. A Simulation Model Representing the OS/VS2 Release 2 Control Program. *Lecture Notes in Computer Science 16.* Springer-Verlag, 1974, 15-29.

[BGS 1982a]
CRYSTAL/IMS Modeling Support Library User's Guide. BGS Systems, Inc., Waltham, MA, 1982.

[BGS 1982b]
CRYSTAL/CICS Modeling Support Library User's Guide. BGS Systems, Inc., Waltham, MA, 1982.

[BGS 1983]
CRYSTAL Release 2.0 User's Guide. BGS Systems, Inc., Waltham, MA, 1983.

[Browne et al. 1975]
J.C. Browne, K.M. Chandy, R.M. Brown, T.W. Keller, D.F. Towsley, and C.W. Dissley. Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems. *Proc. IEEE 63,*4 (June 1975), 966-975.

[Campbell & Heffner 1968]
D.J. Campbell and W.J. Heffner. Measurement and Analysis of Large Operating Systems During System Development. *1968 Fall Joint Computer Conference Proceedings, AFIPS Volume 37* (1968), AFIPS Press, 903-914.

[Lassettre & Scherr 1972]
Edwin R. Lassettre and Allan L. Scherr. Modeling the Performance of the OS/360 Time-Sharing Option (TSO). In Walter Freiberger (ed.), *Statistical Computer Performance Evaluation.* Academic Press, 1972, 57-72.

[Smith 1981]
Connie Smith. Increasing Information Systems Productivity by Software Performance Engineering. *Proc. CMG XII International Conference* (1981).

[Smith & Browne 1982]
Connie Smith and J.C. Browne. Performance Engineering of Software Systems: A Case Study. *1982 National Computer Conference Proceedings, AFIPS Volume 51* (1982), AFIPS Press, 217-244.

[Smith & Browne 1983]
Connie Smith and J.C. Browne. *Performance Engineering of Software Systems: A Design-Based Approach.* To be published, 1983.

[Weleschuk 1981]
B.M. Weleschuk. Designing Operating Systems with Performance in Mind. M.Sc. Thesis, Department of Computer Science, University of Toronto, 1981.