Technical Report
CMU/SEI-95-TR-020
ESC-TR-95-020

State of the Practice Report: Problems in the Practice of Performance Engineering

Mark H. Klein

February 1996

Technical Report

CMU/SEI-95-TR-020
ESC-TR-95-020
February 1996

State of the Practice Report: Problems in the Practice of Performance Engineering

Mark H. Klein

Dynamic Systems Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

Acknowledgments

This report is the product of input received from many performance engineering practitioners and researchers. In particular, this report has benefited from input from Mario Barbacci, Paul Clements, John Goodenough, Larry Howard, Suzie Garcia, David Kelly, Pete Malpass, Ian Pyle, Raj Rajkumar, Tracy Sienknecht, Carol Sledge, and Murray Woodside.

State of the Practice Report:
Problems in the Practice of Performance Engineering

Abstract: All systems have performance requirements, sometimes dominant and explicit, and other times subordinate and implicit. Despite the pervasiveness and importance of performance requirements, performance problems persist. To help us understand why, we sponsored a workshop in performance engineering and conducted some structured interviews with software contractors. This report summarizes our observations.

1 Introduction

1.1 Performance

"*Performance*" has many connotations. The definition of performance given in the IEEE Standard Glossary of Software Engineering Terminology [IEEE-610.12] is "the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage." This is a common connotation, but performance in this report refers to the timeliness aspects of how software systems behave.

"Performance refers to responsiveness: either the time required to respond to specific events or the number of events processed in a given interval of time" [Smith 93]. Performance is that attribute of a computer system that characterizes the timeliness of the service delivered by the system; therefore, *performance engineering* deals with predicting and controlling the timeliness properties of a software engineering artifact.

1.2 Background

All systems have performance requirements. Sometimes performance is fundamental to the correctness of the system; other times performance requirements are implicit and qualitative. Despite the apparent importance of engineering performance into systems and despite the existence of some analytical underpinnings, performance problems are still very common in software-intensive systems. Therefore it is natural to ask:

- What aspects of today's practice contribute to performance problems?
- What are the barriers to achieving a state of practice in which potential performance problems are detected early during the engineering process and avoided?

To investigate these questions we sponsored a workshop in performance engineering. The Second Annual Disciplined Engineering Workshop: *Effective Practice in Performance Engineering* was held on June 28-29, 1995 at the Software Engineering Institute of Carnegie Mellon University in Pittsburgh, Pennsylvania. The intent of the workshop was to provide a forum for interaction and dialogue in performance engineering. We also conducted structured interviews with a small set of software contractors (primarily defense contractors).

1.3 Purpose

The purpose of this report is to call attention to the importance of performance engineering as a neglected discipline, as demonstrated by the information we gathered from the workshop and the interviews. We believe that considerable benefits can be reaped by giving more attention to performance engineering.

Constructive feedback to this report is invited and welcomed. We would like to hear from you if you agree or disagree with the findings. If you agree with the findings and would like to supply more evidence, let us know. Also, we are very interested in ideas for how to mature or transition performance engineering technology. If you disagree with the findings, we would like to know how we can make the report more accurate. The author can be contacted via email at *mk@sei.cmu.edu*.

2 Observations of Performance Engineering Practice

We were able to place the observations discussed below roughly into the following three categories:

- technical development process
- component marketplace
- performance models and measures

Technical development process is concerned with observations about the development and institutionalization of standards of practice. Mature engineering disciplines tend to have mechanisms for codifying practice such as standards of practice which are widely disseminated, regularly reviewed, and updated. Consider structural engineering as an example. The American Association of State Highway and Transportation Officials has written a major design specification for highway bridges that provides standard values for the allowable stresses and loadings for a bridge in this country [Spector 86, p. 272].

Component marketplace is concerned with observations about using, evaluating, standardizing, and combining commercial-off-the-shelf products in the development of systems.

Performance models and measures is concerned with observations about the use of performance models and measures to predict performance, and how trends toward heterogeneous, distributed systems compromise the efficacy of existing methods.

Observations within each of these categories are described in the following sections.

2.1 Technical Development Process

This section is concerned with observations related to the codification of performance engineering into routine software engineering practice. Each observation is described in the following paragraphs and summarized in the italicized text.

Institutionalization of Performance Engineering

The degree to which software performance engineering is institutionalized into the software development process varies considerably between organizations.

There seems to be large disparities between organizations in the degree to which performance engineering is integrated into the software development process. In some organizations that were interviewed by the SEI, performance engineering was recognized as an engineering specialty whose point of view was represented when major architectural decisions were made.[1] All aspects of system performance were within the purview of the perfor-

1. Recorded in an unpublished collection of interview data.

mance team, and the team had veto power over design decisions. Organizations at the other end of the spectrum treated performance merely as an afterthought, the thought occurring only after performance problems were manifest. Oftentimes the separation of the performance analyst and performance methodologies from mainstream activities also inhibited effective integration of performance into the development process.

To respond to a rising pressure of performance problems in medium and large software projects, performance analysis needs to be more uniformly integrated into software development activities.

Standard of Practice

No generally accepted standard of practice exists for performance engineering. However, there are plenty of good sources to draw upon to create one.

All systems have performance requirements; sometimes performance is fundamental to the correctness of the system, as for real-time systems; other times performance requirements are implicit and qualitative. Despite the importance of performance to all systems, there is no reference model or standard of practice against which organizations can compare their own performance engineering practice.

Adoption of performance engineering and its integration into software engineering practice could be facilitated by formalizing and publishing its role. The idea is not to create “the one and only” performance methodology, but to provide concrete guidance with pointers to the best accepted technical practices.

Exemplary performance engineering practice today can be characterized as an iterative process of setting performance goals, developing performance budget allocations, and carrying out performance analysis as discussed by [Smith 90] and [Haskell 95]. Such practice is characterized by the following:

- Performance requirements are quantified as part of system requirements.
- Resource budgets are estimated from experience that is captured in performance databases, from specifications of existing subcomponents, and/or from building prototypes. Budget estimates are then subjected to performance analysis and possibly modified.
- Performance models are developed in concert with system design. As development proceeds, the models are refined and performance measurements and model predictions are mutually corroborating.
- Performance models live beyond the development stage and provide insight into system testing and modification.

Evidence of this type of practice exists. The software engineering community could benefit from the codification and promulgation of today’s exemplary practice, perhaps by the creation of a standard of practice.

Management and Economic Factors

The level of institutionalization of performance engineering within an organization is often linked to the level of management support, which in turn relies on economic justification, which is often lacking.

There are significant economic factors inhibiting the adoption of software performance engineering. When performance engineering is effective, it appears invisible. Management perceives early performance work as an extra cost. Managers need to understand performance engineering investments in terms of risk management and cost avoidance, rather than as engineering overhead. Work is needed to facilitate economic justification for performance engineering. Investment profiles are needed to create expectations about the levels and nature of investment (what the money is used for).

Performance engineering should be viewed as a natural part of the engineering process and managed accordingly. For example:

- Performance data need to be treated as a precious resource and monitored and managed accordingly.
- Performance engineering concepts and practices need to be spread to the lowest levels of the development organization. Individual developers as well as teams need to be cognizant of and concerned with performance engineering.
- Management needs to promote the interaction of developers and customers concerning performance engineering issues: cost, meaning of data, and tradeoffs.

The performance engineering task should be scaled based on an appraisal of cost and risk. If the performance engineering task is minimized or eliminated, it should be a conscious decision based on a cost-benefit analysis using past experience as a guide.

There appears to be a certain degree of organizational infrastructure that is needed to carry out performance engineering as described above. This suggests the possibility of developing links between the Capability Maturity Model^{SM} (CMM^{SM})[2] for Software and performance engineering practices, both as a way of raising awareness of performance engineering and as a way of articulating the relationship between organizational and technical engineering practices.

The lack of perceived value for the performance engineering investment appears to ripple out to other components of the performance engineering community. User demand for vendor-supplied performance information is minimized, and in turn there is little impetus to develop standards for practice or standards for component specifications.

2. Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

2.2 Component Marketplace

This section is concerned with observations related to using, evaluating, standardizing and combining commercial-off-the-shelf (COTS) products in the development of systems.

COTS Performance Evaluation and Criteria

No standard of practice exists for how to evaluate the performance of a software component. As the reliance on COTS increases, the risk of discovering performance problems after it is too late may increase for systems for which performance predictability is critical.

The perceived public benefit for using COTS, combined with increasing COTS availability, makes the problem of COTS evaluation important and pervasive. Currently there is no standard of practice for how to go about evaluating the performance of different classes of COTS products. While benchmarking is common practice, there is no standard for deciding on the most appropriate benchmark or the suitability of existing benchmarks for evaluation.

This lack of guidance makes the build/buy decision more difficult and risky. Choosing a COTS product that has terrible performance can be costly for the purchaser.

A public guide (or standard of practice) for how to evaluate COTS would be helpful to system developers, COTS vendors, and system acquisition agents. Such a guide would set a common framework that would align the expectations of each of the aforementioned parties. For system developers, a guide would provide a checklist of potential risks against which to check the COTS. This would help to ensure that developers do not overlook important performance issues when evaluating a COTS component. A guide could also simplify vendors' evaluation processes. If vendors all evaluate performance using a standard method, they don't have to support customer-specific processes. A standard evaluation practice would also help to ensure that when COTS are written into RFPs they are evaluated against a common standard.

Repository for COTS Usage/Performance Information

There is no objective clearinghouse for performance information for COTS. As a result, the software engineering community does not benefit from its own performance-related experiences in using COTS.

It appears that the performance community does not derive benefits from its own experiences in the use of COTS. While COTS users have information about their experiences which could benefit other users, potential users, and vendors, there is no forum or repository for sharing this information.

Moreover, vendors are not seeing marketplace demand for performance information. In part, this might be because what the vendors supply is not useful or is not trusted. Vendors also appear to be worried that COTS purchasers might be turned off by data that are too complicated, and vendors are worried about the legal ramifications of putting out data whose use is not yet totally understood. Moreover, providing any data is time consuming. As a result, performance issues take a back seat to other issues, and oftentimes COTS components are not upwardly compatible from a performance point of view. New versions of COTS products typically have more functionality and less performance (since more resources are used). Vendors see their market opportunity mainly in terms of functionality, not performance.

Published performance data would make this implicit performance vs. functionality tradeoff much more visible to the COTS consumer community. On one hand, there appears to be a clear disincentive for vendors to provide performance information for new releases. On the other hand, this is exactly the information that users need. Improved dissemination of information about actual use of COTS will benefit the community and increase awareness of performance issues. Examples of useful information include

- typical work-load scenarios
- known performance problems
- hints for performance improvements or solutions to known performance problems
- performance data

One can easily imagine a catalogue of COTS products with such performance information that would be valuable to system developers and have a powerful market influence. Such a catalogue could contain both vendor-supplied information as well as experience-based information collected from the COTS user community. This type of information could be prepared and maintained by a third party. A third-party could

- foster objectivity
- stimulate vendors to agree on reasonable comparison criteria, that is, performance metrics that are both needed by the community and reasonable for vendors to provide
- perhaps eliminate a worry on the part of the vendor that providing such information implies a legal promise

A public guide might involve the development and use of standard COTS usage scenarios for various types of COTS components. Reaching agreement on a typical set of scenarios would greatly facilitate benchmarking, comparison, and optimization of vendor products. While it is typically very hard to identify critical performance cases, scenarios provide a focal point for user-vendor communication, ultimately leading toward improved scenarios and potentially shedding light on how to characterize performance criteria for various classes of components.

This type of service could help to highlight vendors who publish useful metrics. If leading vendors start to publish such data then others might follow, leading to a higher level of confidence in vendor-published data. This, in turn, could lead to developers using and demanding this type of information. Vendors would probably welcome information about how their products are used and misused if such information is required to remain competitive in a performance-aware component marketplace.

Usage/Performance Specifications

There appears to be a need for components to have a "performance interface" for expressing the performance properties of a component and that allows for predictable tuning of the component.

Work needs to be done to develop standard mechanisms for specifying and controlling the performance of components. These mechanisms should

- accommodate varying usage patterns
- accommodate different platforms
- offer guidance for tuning

To accommodate the need for flexibility, vendors oftentimes provide configuration parameters, which can affect performance. Vendor-specified default values are sometimes good enough as a starting point. However, default values are based on the vendor's best guess about anticipated workloads and usage patterns. Naturally, default settings are not appropriate for all individual cases, and vendors should not be expected to know what is best for each user of their products.

However, the user community needs guidance for how to tune components as a function of usage pattern and platform, and vendors need mechanisms for stating how performance varies as a function of the tuning parameters. Typically when performance needs to be improved, there is not enough "interface" information to determine how tuning affects performance. Thus, it is very difficult for the tuner of a COTS component to make sensible choices. One option is to tune the component to perform optimally in isolation. However, local optimizations might not be optimal in the context of the rest of the system. Oftentimes the tuner must rely on his/her own experience with the COTS product, which makes tuning a skilled job that is very dependent on specific individuals. Also, guidelines are needed for how to evaluate the results of changes: what to measure and how to interpret measurements for potential improvements.

While there are issues in the development of standards for performance interface specifications (as stated above), the field of software/computer performance analysis is not new and there is a rich body of experience to draw upon from the real-time systems and the performance modeling communities, among others. See, for example, [Complement 92], [Klein 93], [Lazowska 84], and [Smith 90].

Predicting COTS Performance in Context

COTS users find it difficult to predict and control the interactions between COTS products to achieve overall system performance goals.

COTS products are usually sold to operate within a general-purpose environment, such as an operating system or client-server architecture. Although COTS products may work satisfactorily individually, they often do not work well together.

Embedding COTS components in a larger system can have unpredictable (undesirable) performance caused by undocumented interaction (“covert channels”) between one or more COTS components and the rest of the system. Contention for resources is an example. Resources are used by the COTS product itself and by the environment in which it is embedded; these are not easy to measure separately. Consequently, the same COTS product may perform differently in different environments, and the execution problems of the environment may appear as problems of the COTS product.

Moreover, open distributed systems promote the use of components from multiple vendors with late binding of application-level components. This provides the developer and the user with functional flexibility. However, without systematic and standard mechanisms for describing how multiple components interact, the nonfunctional aspects of the system, such as performance, can render the system unusable.

System performance models that allow one to predict how multiple components interact are needed. Without such models, adjusting parameters to improve one COTS product might degrade others. Also, since limits of usage for different products might be different, it is hard to extrapolate from current use to future use, particularly when behavior is nonlinear and is a function of several effects. System performance models would allow one to factor in the COTS performance data to derive overall system performance.

2.3 Performance Models and Measures

This section is concerned with observations about the use of performance models and measures to predict performance, and how trends toward heterogeneous, distributed systems compromise the efficacy of existing methods.

Simple Conceptual Models

> *Simple conceptual models of performance are needed to make performance engineering more accessible to and usable by the software engineering community.*

Modeling can play an essential role in integrating knowledge of the system and in supporting the exploration of critical issues and alternatives. On the other hand, models that are too complex or too idealized to be useful can be barriers to adoption of performance engineering.

The performance engineering discipline is often perceived as arcane and obscure. For example, formal performance models based on subsystem states, such as Markov Chains or Petri Nets, can result in very complex models which might not give the desired and needed insight into suitable software structures. Moreover, understanding the software implications derivable from theoretical models often requires a deep understanding of the theoretical techniques and their limitations, requiring specialists in modeling.

Modeling becomes increasingly important for complex systems. Without models there is no chance that the complex systems can be understood. Models can be used to

- integrate knowledge of the various parts of a system
- identify gaps in knowledge
- manage complexity by highlighting the most relevant details and hiding less important details
- inform architectural decisions by helping to identify trouble spots early in the life cycle and enable affordable investigation of alternatives

Performance models (and any models for that matter) offer insight into how to reason about the performance ramifications of architectural decisions. They should serve as a useful abstraction of the real system by providing performance predictions for systems early in development and by predicting the consequences of change in a manner that is easier than changing the system itself. Models should be maintained with the design throughout the life cycle, and used to explain and justify the design from a performance point of view.

Adoption of performance engineering and its integration into software engineering will be facilitated by the generation of a simplified conceptual model for performance. This model should foster communication between the performance specialist and others, and make performance engineering more accessible to the nonspecialist. Different aspects of performance models are discussed, for example, in [Woodside 95a], [Smith 90], and [Chatterjee95[3]].

Heterogeneity and Complexity

Heterogeneity and complexity are compromising the ability to understand and model system performance.

Heterogeneity is a common feature of today’s large, complex distributed systems. Heterogeneity has many aspects including hardware, operating systems, languages, and protocols. Moreover, each of these aspects has its own trajectory of evolution. Heterogeneity drives the cost of modeling up by requiring the connection of multiple, different models and model elements together. Heterogeneity also increases the cost of measurement since different types and kinds of instrumentation are required, and the correlation of measurement data is required.

Another cause of complexity that challenges current modeling capabilities is the large number and types of interactions between inadequately defined components. Physically parallel parts of the system running on separate nodes can interact, making a global system with a large state space for analysis. Temporal dependencies can add further complexity. Such systems can exhibit many different kinds of behavior with intricate conditions for triggering each type of behavior.

3. Chatterjee, S. and Strosnider, J., “Distributed Pipeline Scheduling: A Framework for Distributed, Heterogeneous Real-Time System Design”, to appear in *The Computer Journal*.

Some parts of the system can be outside of the visibility of the system designer. For example, client/server architectures and object-orientation can hide details that are important for performance modeling. Thus, abstraction from one point of view can introduce complexity from another point of view. Also, the behavior of other systems with which a system interacts (including humans and other computer systems) might not be totally known and thus compromise the ability to predict performance.

Some of the aforementioned aspects of complexity and heterogeneity defeat well-established modeling techniques such as queueing-based techniques. For instance, mechanisms involving the simultaneous use of logical and physical resources, and behavior patterns involving interactions between separate processes (either concurrent in the same node, or distributed on a network) require complicated extensions to standard models [Smith 90, Section 8.2], [Agrawal 83]. Viable techniques will rely on approximations that are either still being developed or are not mature enough for most software practitioners to use. Examples of such approximations can be found in the research literature in database system modeling [Thomasian 89]; in layered system models for distributed systems [Woodside 95a], [Rolia 95]; and in models of parallel processing [Kapelnikov 89].

Existing modeling techniques are not used to their fullest potential. Many commercial tools tend to be complex, difficult to learn, and very detail oriented. In addition, they are costly to use and are not widely accepted. (See [Smith 90],[Smith 95] for a list of commercially available performance modeling tools.) Training for software engineers in modeling and modeling theory is generally lacking.

Architecture-Level Performance Analysis

Architecture reviews require more emphasis on performance. Performance engineering issues need to become a standard part of architecture reviews.

Performance engineering issues should become a normal part of an architecture-level review. Current architecture reviews do not put sufficient emphasis on predicting performance of the system, resulting in surprises when the system is built. Since architectural flaws which hurt performance can be very hard to remove, architecture review is an appropriate time for the results of up-front performance engineering work to be presented and reviewed. Performance guidance is needed early in the development process since every decision potentially reduces the degrees of freedom in how to achieve effective performance. Architecture review is an opportunity to recognize and (re)use patterns with known performance properties and to compare the proposed architecture with patterns used on similar and/or dissimilar systems.

Performance Measurement

Measuring heterogeneous, complex systems is difficult and costly. Measurement needs to be more closely linked to underlying models, and tools are needed to support this.

Measurement is fundamental to all engineering disciplines: it is necessary to understand the parameters of importance and their relationships, and to have tools and techniques for obtaining measurements. Measurements must be linked back to models to facilitate data interpretation and to guide measurement. Measurements are more useful in the context of a model; otherwise, it is easy to sink in a sea of measurement data.

In heterogeneous environments, there can be problems correlating data acquired with measurement tools that are specific to different portions of the system. For example, when the network instruments are separate from the computer measurement instruments, there often is no common time base for identifying events. When a message is sent, it can be seen in the computer and on the network, but there is no way to tell if it is the same message.

Another measurement problem is that most measurement tools do not record the software context of the execution; this is an old problem with hardware instrumentation of computers. Apparently it is still difficult to relate measurement data to a source-code context. One cause of this is that advances in technology are making it difficult or impossible to place measurement probes. For example, on-chip cache makes current address lines and instructions invisible.

Granularity of time measurement is a problem originating from different clock speeds in different computers. Even for different platforms that use the same hardware there can be differences of precision of several orders of magnitude.

COTS products also present special difficulties as mentioned in the previous section discussing COTS. Measurement interfaces for COTS components will be helpful and necessary.

3 Conclusions

This report documents observations about areas that need to be improved in performance engineering.

Currently, there appears to be a large spectrum of performance engineering practice. A codification of best practice in a “standard of practice” in a yet-to-be-determined form could be quite beneficial to the software engineering community. The standard should include guidance for how to consider performance during software architecture development and review. Linking the standard of practice to the Capability Maturity Model for Software [Paulk 93] could be one mechanism for promulgating it and encouraging its usage. Other mechanisms include professional societies and standards organizations.

The trend toward heterogeneous distributed systems built in part from COTS components exacerbates the performance engineering problem and to some degree impedes performance engineering. Several observations from [Allen 77] seem applicable:

> *It is becoming generally accepted that technology builds upon itself and advances quite independently of any link with the scientific frontier, and often without any necessity for an understanding of the basic science which underlies it.*
>
> *Occasionally, technology is forced to forfeit some of its independence. This happens when its advance is impeded by a lack of understanding of the scientific basis of the phenomena with which it is dealing.*

Continued work is needed in applying performance modeling techniques to understand performance in heterogeneous, distributed settings. Continued work is also needed to understand how to better characterize the performance of COTS components and how to use such characterizations in concert with performance models to predict system performance.

Finally, it should be noted that the observations in this report are from a limited sample; nevertheless, a broad spectrum of the software engineering community is represented. This leads us to believe that our observations are representative of the problems in practice of performance engineering. However, continued feedback will be welcomed.

Bibliography

Agrawal 83	Agrawal, S.C. & Buzen, J.P. "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems." *ACM Trans. on Computer Systems 1*, 2 (May 1983): 116-143.
Allen 77	Allen, T.J., *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information Within the R&D Organization*. Cambridge, MA: MIT Press, 1977.
Complement 92	Ayache, S.; Conquet, E. & Puigjaner, R., “Taxonomy of Performance Requirements," *Proceedings of the Complement Consortium,* Ref TFP7-1.0. January 1992.
Haskell 95	Haskell, Lou. "A Performance Engineering Approach." *Second Annual Disciplined Engineering Workshop: Effective Practice in Performance Engineering*. Pittsburgh, PA, June 28-19, 1995.
IEEE-610.12	Institute of Electrical and Electronic Engineers. "IEEE Standard Glossary of Software Engineering Terminology." *IEEE Standards Collection.* New York, NY: Institute of Electrical and Electronics Engineers, 1993.
Kapelnikov 89	Kapelnikov, A.; Muntz, R. R. & Ercegovac, M. D., “A Modelling Methodology for the Analysis of Concurrent Systems and Computations," *J. Parallel Distributed Comput. 6*, (1989): 568-597.
Klein 93	Klein, M.H.; Ralya, T.; Pollack, B.; Obenza, R. & Harbour, Michael Gonzalez. *A Practitioners’ Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993.
Lazowska 84	Lazowska, E. D; Zahorjan, J.; Graham, G. S.; & Sevik, K. C. *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1984

Paulk 93	Paulk, M.; Curtis, B.; Chrissis, M. & Weber, C., *Capability Maturity Model for Software Version 1.1*, (CMU/SEI-93-TR-24, ADA 963403.). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.
Rolia 95	Rolia, J.A. & Sevcik, K.C. “The Method of Layers." *IEEE Trans. on Software Engineering 21*, 8 (August 1995): 689-700.
Smith 90	Smith, C. U. *Performance Engineering of Software Systems*, The SEI Series in Software Engineering. Reading, MA: Addison-Wesley Publishing Comp., 1990.
Smith 93	Smith, C. U. & Williams, L. G. “Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives,” *IEEE Transactions on Software Engineering 19*, 7 (July 1993): 720-741.
Smith 95	Smith, C. U. & Williams, L. G., “A Performance Model Interchange Format,” *Performance Engineering Services*, Feb. 1995.
Spector 86	Spector, A. & Gifford, D. “A Computer Science Perspective of Bridge Design.” *Communications of the ACM 29*, 4 (April 1986): 267-283.
Stankovic 88	Stankovic, J. A. “Misconceptions About Real-Time Computing: A serious problem for next-generation systems.” *IEEE Computer 21*, 10 (October 1988): 10-19.
Thomasian 89	Thomasian, A. & In Kyung Ryu. "A Recursive Solution Method to Analyze the Performance of Static Locking Systems." *IEEE Trans. on Software Eng.15*, 10 (October 1989): 1147-1156.
Woodside 95a	Woodside, C.W.; Neilson, J. E.; Petriu, D. C.; & Majumdar, S. “The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software." *IEEE Transactions on Computer 44*, 1 (January 1995): 20-34.

Woodside 95b	Woodside, C.M. “A Three-View Model for Performance Engineering of Concurrent Software." *IEEE Transactions on Software Engineering 21*, 9 (September 1995): 754-767.