The Business Case for Software Performance Engineering

Lloyd G. Williams, Ph.D.
Connie U. Smith, Ph.D.

March, 2002

Contents

Executive Summary

Performance failures occur when a software product is unable to meet its overall objectives due to inadequate performance. Performance failures negatively impact your bottom line by increasing costs, decreasing revenue or both. This white paper discusses the cause of performance failures and shows how they can be cost-effectively prevented.

Performance Failures: Cause and Prevention The primary cause of performance failures is a reactive approach to performance during the development process. Cost and schedule pressures encourage project managers to adopt a "fix-it-later" approach in which performance is ignored until there is a problem. When a problem is discovered, you must buy more hardware, developers must try to "tune" the software to meet performance objectives or both, causing schedule delays and cost overruns. In some cases, it is simply not possible to meet performance objectives by tuning.

Performance problems are most often due to inappropriate architectural choices rather than inefficient coding. This means that performance problems are introduced early in the development process but are typically not found until later (during integration test or when the system is in use) when they are more difficult and more expensive to fix.

The key to preventing performance failures and the resulting project crises is to adopt a proactive approach to performance management that anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. With a proactive approach, you produce software that meets performance objectives and is delivered on time and within budget, and avoid the project crises brought about by discovering performance problems late.

Software Performance Engineering Software performance engineering (SPE) is a systematic, quantitative approach to proactively managing software performance. SPE is an engineering approach that avoids the extremes of performance-driven development and "fix-it-later." With SPE, you detect problems early in development, and use quantitative methods to support cost-benefit analysis of various solution options. SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. It uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. It also includes techniques for collecting data, principles and patterns for performance-oriented design, and anti-patterns for recognizing and correcting common performance problems.

Costs and Benefits The cost of using SPE is usually a small percentage of the overall project budget. The level of SPE effort is determined by the amount of performance risk on the project. For a project with little risk, SPE costs are typically 1% of the total budget. For a high-risk project, the SPE effort might be as high as 10% of the project budget.

Industrial experience indicates that SPE can save far more than it costs by avoiding the costs (both direct and indirect) of performance failure mentioned above.

Getting Started Begin with a single project. Train key developers in the SPE process and techniques and provide then with the necessary tool support. Then, use the experience gained on that project to make SPE and integral part of the software development process on your other projects.

Introduction

Performance failures occur when a software product is unable to meet its overall objectives due to inadequate performance. Performance failures negatively impact your bottom line by increasing costs, decreasing revenue or both. This white paper discusses the cause of performance failures and shows how they can be cost-effectively prevented.

The Importance of Software Performance

Performance is any characteristic of a software product that you could, in principle, measure by sitting at the computer with a stopwatch in your hand. The dimensions of performance include responsiveness (response time or throughput) and scalability.

Performance is an essential quality attribute of every software system. Many software systems, however, cannot be used as they are initially implemented due to performance problems. The following anecdote is typical.

> NASA was forced to delay the launch of a satellite for at least eight months. The satellite and the Flight Operations Segment (FOS) software running it are a key component of the multibillion-dollar Earth Science Enterprise, an international research effort to study the interdependence of the Earth's ecosystems. The delay was caused because the FOS software had unacceptable response times for developing satellite schedules, and poor performance in analyzing satellite status and telemetry data. There were also problems with the implementation of a control language used to automate operations. The cost of this rework and the resulting delay has not yet been determined. Nevertheless it is clearly significant, and the high visibility and bad press is potentially damaging to the overall mission. Members of Congress also questioned NASA's ability to manage the program. [Harreld 1998a], [Harreld 1998b]

This anecdote illustrates a "performance failure"—the inability of a software product to meet its overall objectives due to inadequate performance.

The Cost of Performance Failures

As the anecdote above illustrates, poor performance can be costly. The costs of performance failures are both direct and indirect. Direct costs incurred due to performance failures include:

- *Increased Operational Costs*—Poor performance means that your staff needs more time to complete key tasks, or that you need more staff to complete these tasks in the same amount of time. In extreme cases, users may bypass the automated system altogether in favor of faster manual processes.
- *Increased Development Costs*—One company discovered, during integration testing, that an online transaction that should have taken 10 seconds could not be completed in less that 60 seconds. Some transactions took as long as 200 seconds. When problems like these arise, you need to allocate additional resources to the project to "tune" or even redesign the software to try to meet performance objectives.
- *Increased Hardware Costs*—If tuning or redesign isn't sufficient to solve the problem, you may need to increase your hardware capacity (for example, by adding more processors or upgrading to faster disks) to achieve your performance objectives.
- *Canceled Projects*—In some cases it will be impossible to meet performance objectives by tuning, and too expensive to redesign the system late in the process or add more hard-

ware capacity. These projects will be canceled and their costs will be largely unrecoverable.

Indirect costs of performance failures include:

- *Damaged Customer Relations*—Poorly performing software can cause your organization's image to suffer. The effects of poorly performing Web sites are well documented; customers will simply go elsewhere rather than endure long waits. This problem is not limited to Web sites, however. Long waits on the telephone while customer-service representatives access customer data will ultimately have the same effect. Even if the problem is fixed later, negative perceptions will continue.
- *Lost Income*—"Tuning" or redesign results in late deployment or delivery of software. In some cases, you may find yourself paying penalties for late delivery or failure to meet contractual performance requirements.
- *Reduced Competitiveness*—Late delivery due to "tuning" or redesign can also mean that you miss a critical market window, allowing your competition to increase their market share at your expense.

The Cause of Performance Failures

The primary cause of performance failures is a reactive approach to performance during the development process. Today's cost restrictions and short development times often encourage project managers to adopt a "Make it run, make it run right, make it run fast" approach. This "fix-it-later" attitude can be dangerous. Performance problems are most often due to inappropriate architectural choices rather than inefficient coding. This means that performance problems are introduced early in the development process but are typically not found until later (during integration test or when the system is in use) when they are more difficult and more expensive to fix. When problems are found, the project usually goes into crisis mode to try and meet performance objectives.

Preventing Performance Failures

The best way to prevent performance failures is to adopt a proactive approach to managing performance during development. A proactive approach anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. By adopting a proactive approach to performance, you avoid the project crises brought about by discovering problems late and produce software that meets performance objectives and is delivered on time and within budget.

The following anecdotes illustrate how using today's best practices to proactively manage performance can improve software quality and prevent performance failures without the need to resort to last-minute heroics.

> An airline reservation service bureau revised its airfare quote system to improve the accuracy of the "lowest fare" quotes. Performance engineers worked closely with developers throughout the project. The result was a system with 100 percent accurate quotes and *improved* performance.
>
> A major insurance company designed a system to provide Web access for its own agents as well as independent agents. The first version of the design called for a large amount of code (in the form of ActiveX agents) to be downloaded to client machines. Performance models of this approach showed that, if the downloaded code underwent a significant upgrade, it would take approximately

three days at full bandwidth to download the changes to all of the client machines. The design was changed to rely less on downloaded code, and the system was deployed successfully.

Tuning efforts to correct problems that arise due to a failure to proactively manage performance often masquerade as "successes." While tuning a system that is in trouble can produce noticeable improvements, the resulting performance is unlikely to equal that of a system for which performance has been designed-in from the beginning. In addition, tuning at the end of the project will invariably be more costly and time-consuming than doing it right the first time.

Software Performance Engineering

Software performance engineering (SPE) [Smith 1990], [Smith and Williams 2002] is a systematic, quantitative approach to constructing software systems that meet performance objectives. SPE is an engineering approach to performance, avoiding the extremes of performance-driven development and "fix-it-later." With SPE, you detect problems early in development, and use quantitative methods to support cost-benefit analysis of hardware solutions versus software requirements or design solutions, versus a combination of the two. You implement software solutions before problems are manifested in code; organizations implement hardware solutions before testing begins. The quantitative assessment identifies trade-offs in software functions, hardware size, quality of results, and resource requirements.

SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. It uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. The models assist developers in controlling resource requirements by enabling them to select architecture and design alternatives with acceptable performance characteristics. The models aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle (typically during final testing).

SPE also prescribes principles and performance patterns for creating responsive software, performance antipatterns for recognizing and correcting common problems, the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis methods.

Aren't We Already Doing That?

Do you have—or can you get—precise, quantitative answers to the following questions *before* your developers begin coding on a project?

1. Will your processing complete in the allotted time? What is the projected response or turnaround time?
2. Are your hardware and software capable of supporting the load? What percentage of the available resources is it projected to take?
3. How well does the architecture support your current performance goals and meet your future scalability goals? What are the projected response or turnaround times for the forecast workload volume over the years in the planning horizon?
4. Do you have a diagram that shows the end-to-end steps in key business tasks and the projected amount of time required for each?

If not, it is likely that you are not already doing SPE.

The fact that you have “performance specialists” on your staff does not guarantee that you are using SPE techniques. In most organizations, performance specialists are oriented toward system support. They typically focus on tuning operating system and middleware parameters rather than the application software. Many are unaware of application-level performance problems and their solution.

It is possible that your developers may, in fact, be aware of SPE techniques (many are not—it is taught in only a few universities). Nevertheless, they may feel that they do not have the time to apply the techniques within the short time frames in most project schedules. Developers often have the attitude that “I’d like to use SPE, but my managers will not allocate enough time in the schedule.”

If you outsource all or part of your software development, you may assume that the vendor is using SPE. Some outsourcing organizations do regularly use SPE techniques. They build it into their bid, cost, and schedule. Others feel pressure to produce low bids or short schedules to get the job, and thus omit these extra steps—especially if they are not legally bound to meet specific performance requirements, or to conduct the SPE steps. The bottom line is: if it’s not in the contract, it’s probably not happening.

What Does It Cost?

The cost of using SPE to proactively manage software performance depends on the size and complexity of the system under development, the level of performance risk, and the expertise and experience of the development team, as well as other factors. If not meeting your performance goals would endanger the success of your project, you have a performance risk. Factors that increase performance risk include: the use of new technologies, lack of experience in the application area, schedule, market factors, and others.

SPE is a risk-driven process. The level of risk determines the amount of effort that you put into SPE activities. If the level of risk is small, the SPE effort can be correspondingly small. If the risk is high, then a more significant SPE effort is needed. For a low-risk project, the cost of the SPE effort required is typically about 1% of the total project budget. For high-risk projects, the SPE effort might be as high as 10% of the project budget.

Return on Investment

When you apply SPE successfully, you are preventing problems. Thus, calculating a return on investment requires tracking or estimating the savings that you realize from avoiding these problems as well as the costs of applying SPE.

Experience reports from industry indicate that the use of SPE to proactively manage performance can save far more than it costs. For example, Banc One Services Corporation reported that, on one project, SPE costs over a five-month period were $147,000. During this time, the team analyzed three applications and identified modifications that resulted in a projected *annual* savings (cost avoidance) of $1,300,000 [Manhardt 1998]. Similarly, the performance engineering group at MCI reported a $20,000,000 savings in one year with SPE due to reduced resource requirements that resulted in deferred configuration upgrades [CMG 1991].

Getting Started

Begin by taking stock of your current situation. Assign a “key technology specialist” to investigate SPE, whether it is being applied on specific projects (and the outcome), whether projects that experienced severe performance problems upon deployment used SPE techniques, and whether it is a standard part of your development process. If so, you are in great shape, but it’s important to make sure that your SPE program is well-integrated into your development process and not dependent on a key person who may leave the company.

If you are not already doing SPE, begin with a single project. Train key developers in the SPE process and techniques and provide them with the necessary tool support. Then, use the experience gained on that project to make SPE an integral part of the software development process on your other projects.

For More Information

For more information contact:

Software Engineering Research
264 Ridgeview Lane
Boulder, CO 80302
(303)938-9847
FAX: (303) 443-5279
boulderlgw@aol.com

Performance Engineering Services
PO Box 2640
Santa Fe, NM 87504
(505) 988-3811
FAX: (786) 513-0165
cusmith@perfeng.com
www.perfeng.com

More information about SPE is also in the book *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software* [Smith and Williams 2002].

Summary

Performance failures occur when a software product is unable to meet its overall objectives due to inadequate performance. Performance failures negatively impact your bottom line by increasing costs, decreasing revenue or both.

The primary cause of performance failures is a reactive approach to performance during the development process. Cost and schedule pressures encourage project managers to adopt a “fix-it-later” approach in which performance is ignored until there is a problem. When a problem is discovered, developers must try to “tune” the software to meet performance objectives if, indeed, they can be met.

The key to preventing performance failures and the resulting project crises is to adopt a proactive approach to performance management that anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. With a proactive approach, you avoid the project crises brought about by discovering performance problems late and produce software that meets performance objectives and is delivered on time and within budget.

Software performance engineering (SPE) is a systematic, quantitative approach to proactively constructing software systems that meet performance objectives. SPE is an engineering approach to performance, avoiding the extremes of performance-driven development and "fix-it-later." With SPE, you detect problems early in development, and use quantitative methods to support cost-benefit analysis of hardware solutions versus software requirements or design solutions, versus a combination of the two.

SPE is a cost effective approach to managing software performance. The level of SPE effort is determined by the amount of performance risk on the project. For a project with little risk, SPE costs are typically 1% of the total budget. For a high-risk project, the SPE effort might be as high as 10% of the project budget. Experience indicates that the use of SPE to proactively manage performance can save far more than it costs.

References

[CMG 1991] Computer Measurement Group, Software Performance Engineering Panel, moderator C. U. Smith, Computer Measurement Group, December, 1991.

[Harreld 1998a] H. Harreld, "NASA Delays Satellite Launch After Finding Bugs in Software Program," *Federal Computer Week*, April 20, 1998.

[Harreld 1998b] H. Harreld, "Panel Slams EOSDIS," *Federal Computer Week*, September 14, 1998.

[Manhardt 1998] D. Manhardt, "Applications Optimization Methodology—An Approach," *Proceedings, First International Workshop on Software and Performance*, Santa Fe, NM, pp. 93-100, October 1998.

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

[Smith and Williams 2002] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Boston, MA, Addison-Wesley, 2002.

About the Authors

Dr. Lloyd G. Williams is a principal consultant at Software Engineering Research, where he specializes in the development and evaluation of software architectures to meet quality objectives, including performance, reliability, modifiability, and reusability. His experience includes work on systems in fields such as process control, avionics, telecommunications, electronic funds transfer, Web-based systems, software development tools and environments, and medical instrumentation. Dr. Williams has been a pioneer in the application of Software Performance Engineering (SPE) to object-oriented systems. He is the author of numerous technical papers and has presented professional development seminars and consulted on software development for more than 100 organizations worldwide.

Dr. Connie U. Smith a principal consultant of the Performance Engineering Services Division of L&S Computer Technology, Inc., is known for her work in defining the field of SPE and integrating SPE into the development of new software systems. Dr. Smith received the Computer Measurement Group's prestigious AA Michelson Award for technical excellence and professional contributions for her SPE work. She also authored the original SPE book: Performance Engineering of Software Systems, published in 1990 by Addison-Wesley, and approximately 100 scientific papers. She is the creator of the *SPE•ED*™ performance engineering tool. She has over 25 years of experience in the practice, research and development of the SPE performance prediction techniques.

Together, Drs. Williams and Smith have over 50 years of experience in software development. They have worked together for more than 15 years to help clients design and implement software that meets performance objectives. They have published numerous technical papers and articles, and are the authors of *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, published by Addison-Wesley.