Best Practices for Software Performance Engineering

Connie U. Smith, Ph.D.
Performance Engineering Services
PO Box 2640
Santa Fe, New Mexico, 87504-2640
(505) 988-3811
http://www.perfeng.com/

Lloyd G. Williams, Ph.D.
Software Engineering Research
264 Ridgeview Lane
Boulder, Colorado 80302
(303) 938-9847
boulderlgw@aol.com

Performance—responsiveness and scalability—is a make-or-break quality for software. Software Performance Engineering (SPE) provides a systematic, quantitative approach to constructing software systems that meet performance objectives.It prescribes ways to build performance into new systems rather than try to fix them later. Many companies successfully apply SPE and they attest to the financial, quality, customer satisfaction and other benefits of doing it right the first time.

This paper describes 24 best practices for applying SPE to proactively managing the performance of new applications. They are vital for successful, proactive SPE efforts, and they are among the practices of world-class SPE organizations. They will help you to establish new SPE programs and fine tune existing efforts in line with practices used by the best software development projects.

1.0 Introduction

Performance—responsiveness and scalability—is a make-or-break quality for software. Software performance engineering (SPE) [Smith and Williams 2002], [Smith 1990] provides a systematic, quantitative approach to constructing software systems that meet performance objectives. 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, or a combination of software and hardware solutions.

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.

This paper presents 24 “best practices” for SPE in four categories: project management, performance modeling, performance measurement, and techniques. A best practice is:

> “a process, technique, or innovative use of technology, equipment or resources that has a proven record of success in providing significant improvement in cost, schedule, quality, performance, safety, environment, or other measurable factors which impact an organization.” [Javelin 2002]

The best practices presented here are based on:

- observations of companies that are successfully applying SPE,
- interviews and discussions with practitioners in those companies, and
- our own experience in applying SPE techniques on a variety of consulting assignments.

Many of them can be found in the *Performance Solutions* book [Smith and Williams 2002]. Ten of them were presented in [Smith and Williams 2003a]. This paper builds on the earlier paper, and puts them in the four categories.

These best practices represent documented strategies and tactics employed by highly admired companies to manage software performance. They have imple-

mented these practices and refined their use to place themselves and their practitioners among the best in the business for their ability to deliver software that meets performance objectives and is on-time and within budget.

2.0 Project Management Best Practices

These are practices adopted by managers of software development projects and/or managers of SPE specialists who work with development managers.

2.1 Perform An Early Estimate Of Performance Risk

It is important to understand your level of performance risk. A risk is anything that has the possibility of endangering the success of the project. Risks include: the use of new technologies, the ability of the architecture to accommodate changes or evolution, market factors, schedule, and others.

If failing to meet your performance goals would endanger the success of your project, you have a performance risk. If your project supports a critical business function and/or will be deployed with high visibility (such as a key, widely publicized web application), then failing to meet performance objectives may result in a business failure and you have an extreme performance risk. Inexperienced developers, lack of familiarity with the technology, a cutting-edge application, and aggressive schedule all increase your risk of performance failure.

To assess the level of performance risk, begin by identifying potential risks. You will find an overview of software risk assessment and control in [Boehm 1991]. Once you have identified potential risks, try to determine their impact. The impact of a risk has two components: its probability of happening, and the severity of the damage that would occur if it did. For example, if a customer were unable to access a Web site within the required time, the damage to the business might be extreme. However, it may also be that the team has implemented several similar systems, so the probability of this happening might be very small. Thus, the impact of this risk might be classified as moderate. If there are multiple performance risks, ranking them according to their anticipated impact will help you address them systematically.

2.2 Match The Level of SPE Effort To The Performance Risk

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 amount of SPE effort required might be about 1% of the total project budget. For high-risk projects, the SPE effort might be as high as 10% of the project budget.

2.3 Track SPE Costs And Benefits

Successful application of SPE is often invisible. If you are successfully managing performance, you do not have performance problems. Because of this, it is necessary to continually justify your SPE efforts. In fact, we have heard managers ask “Why do we have performance engineers if we don’t have performance problems?”

It is important to track the costs and benefits of applying SPE so that you can document its financial value and justify continued efforts. The costs of SPE include salaries for performance specialists, tools, and support equipment such as workstations for performance analysts or a dedicated performance testing facility. The benefits are usually costs due to poor performance that you reduce or avoid as a result of applying SPE. These include: costs of refactoring or tuning, contractual penalties, user support costs and lost revenue as well as intangible costs such as damaged customer relations.

Once you have this information, it is easy to calculate the return on investment (ROI) [Reifer 2002] for your SPE efforts. The return on investment for SPE is typically more than high enough to justify its continued use (see, for example, [Williams, et al. 2002] and [Williams and Smith 2003b]).

2.4 Integrate SPE Into Your Software Development Process and Project Schedule

To be effective, SPE should not be an “add-on;” it should be an integral part of the way in which you approach software development. Integrating SPE into the software process avoids two problems that we have seen repeatedly in our consulting practice. One is over-reliance on individuals. When you rely on individuals to perform certain tasks instead of making them part of the process, those tasks are frequently forgotten when those individuals move to a different project or leave the company.

The second reason for making SPE an integral part of your software process is that many projects fall behind schedule during development. Because performance problems are not always apparent, managers or developers may be tempted to omit SPE studies in favor of meeting milestones. If SPE milestones are defined and enforced, it is more difficult to omit them.

2.5 Establish Precise, Quantitative Performance Objectives And Hold Developers and Managers Accountable For Meeting Them

Precise, quantitative performance objectives help you to control performance by explicitly stating the required performance in a format that is rigorous enough so that you can quantitatively determine whether the software meets that objective. Well-defined performance objectives also help you evaluate architectural and design alternatives and trade-offs and select the best way of meeting performance (and other quality) requirements.

It is important to define one or more performance objectives for each performance scenario. Throughout the modeling process, you can compare model results to the objective, to determine if there is significant risk of failing to meet the objectives, and take appropriate action early. And, as soon as you can get measurements from a performance test, you can determine whether or not the software meets the objective.

A well-defined performance objective would be something like: "The end-to-end time for completion of a 'typical' correct ATM withdrawal performance scenario must be less than 1 minute, and a screen result must be presented to the user within 1 second of the user's input." Vague statements such as "The system must be efficient" or "The system shall be fast" are not useful as performance objectives.

For some types of systems you may define different performance objectives, depending on the intensity of requests. For example, the response time objective for a customer service application may be 1 second with up to 500 users or less, 2 seconds for 500 to 750 users, and 3 seconds for up to 1,000 users.

Unless performance objectives are clearly defined, it is unlikely that they will be met. In fact, establishing specific, quantitative, measurable performance objectives is so central to the SPE process that we have made it one of the performance principles [Smith and Williams 2002]. When a team is accountable for, and rewarded for achieving their system's performance, they are more likely to manage it effectively. If the team is only accountable for completion time and budget, there is no incentive to spend time or money for performance.

2.6 Identify Critical Use Cases And Focus On The Scenarios That Are Important To Performance

Use cases describe categories of behavior of a system or one of its subsystems. They capture the user's view of what the system is supposed to do. Critical use cases are those that are important to responsiveness as seen by users, or those for which there is a performance risk. That is, critical use cases are those for which the system will fail, or be less than successful, if performance goals are not met.

Not every use case will be critical to performance. The 80-20 rule applies here: A small subset of the use cases ($\leq$20%) accounts for most of the uses ($\geq$80%) of the system. The performance of the system is dominated by these heavily used functions. Thus, these should be your first concern when assessing performance.

Don't overlook important functions that are used infrequently but must perform adequately when they are needed. An example of an infrequently used function whose performance is important is recovery after some failure or outage. While this may not occur often, it may be critical that it be done quickly.

Each use case is described by a set of scenarios that describe the sequence of actions required to execute the use case. Not all of these scenarios will be important from a performance perspective. For example, variants are unlikely to be executed frequently and, thus, will not contribute significantly to overall performance.

For each critical use case, focus on the scenarios that are executed frequently, and on those that are critical to the user's perception of performance. For some systems, it may also be important to include scenarios that are not executed frequently, but whose performance is critical when they are executed, such as recovery from an outage.

Select the scenarios, *get consensus* that they are the most important, then focus on their design and implementation to expedite processing and thus optimize their responsiveness. People are more likely to have confidence in the model results if they agree that the scenarios used and workloads used to obtain the results are representative of those that are actually likely to occur. Otherwise, it is easy to rationalize that any poor performance predicted by the models is unlikely, because the performance scenarios chosen will not be the dominant workload functions. The scenarios also drive the measurement studies by specifying the conditions that should be performance tested.

2.7 Perform an Architecture Assessment to Ensure That the Software Architecture Will Support Performance Objectives

Recent interest in software architectures has underscored the importance of architecture in determining software quality. While decisions made at every phase of the development process are important, architectural decisions have the greatest impact on quality attributes such as modifiability, reusability, reliability, and perfor-

mance. As Clements and Northrop note [Clements and Northrop 1996]:

> "Whether or not a system will be able to exhibit its desired (or required) quality attributes is largely determined by the time the architecture is chosen."

While a good architecture cannot guarantee attainment of performance objectives, a poor architecture can prevent their achievement.

Architectural decisions are among the earliest made in a software development project. They are also the most costly to fix if, when the software is completed, the architecture is found to be inappropriate for meeting quality objectives. Thus, it is important to be able to assess the impact of architectural decisions on quality objectives such as performance and reliability at the time that they are made.

Performance cannot be retrofitted into an architecture without significant rework; it must be designed into software from the beginning. Thus, if performance is important, it is vital to spend the up-front time necessary to ensure that the architecture will not hinder attainment of performance requirements. The "make it run, make it run right, make it run fast" approach is dangerous. Our experience is that performance problems are most often due to inappropriate architectural choices rather than inefficient coding. By the time the architecture is fixed, it may be too late to achieve adequate performance by tuning.

The method that we use for assessing the performance of software architectures is known as PASA^{SM} [Williams and Smith 2002]. It was developed from our experience in conducting performance assessments of software architectures in a variety of application domains including web-based systems, financial applications, and real-time systems. PASA uses the principles and techniques of software performance engineering (SPE) to determine whether an architecture is capable of supporting its performance objectives. The method may be applied to new development to uncover potential problems when they are easier and less expensive to fix. It may also be used when upgrading legacy systems to decide whether to continue to commit resources to the current architecture or migrate to a new one.

2.8 Secure The Commitment To SPE At All Levels Of The Organization

The successful adoption of SPE requires commitment at all levels of the organization. This is typically not a problem with developers. Developers are usually anxious to do whatever is needed to improve the quality of their software.

If there is a problem with commitment, it usually comes from middle managers who are constantly faced with satisfying many conflicting goals. They must continually weigh schedule and cost against quality of service benefits. Without a strong commitment from middle managers, these other concerns are likely to force SPE aside. Commitment from upper management is necessary to help middle managers resolve these conflicting goals.

2.9 Establish an SPE Center of Excellence to Work with Performance Engineers on Project Teams

It is important that you designate one or more individuals to be responsible for performance engineering. You are unlikely to be successful without a performance engineer (or a performance manager) who is responsible for:

- Tracking and communication of performance issues
- Establishing a process for identifying and responding to situations that jeopardize the attainment of the performance objectives
- Assisting team members with SPE tasks
- Formulating a risk management plan based on shortfall and activity costs
- Ensuring that SPE tasks are properly performed

The responsible person should be high enough in the organization to cause changes when they are necessary. The performance engineering manager should report either to the project manager or to that person's manager.

The person responsible for performance engineering should be in the development organization rather than the operations organization. You will have problems if responsibility for SPE is in the operations organization because developers will likely put priority on meeting schedules over making changes to reduce operational costs.

Making SPE a function of the capacity planning group is also a mistake in most organizations, even though that group usually already employs individuals with performance modeling expertise. While some capacity planners have the performance engineering skills, most are mathematical experts who are too far removed from the software issues to be effective.

With the "SPE Center of Excellence" approach, members of the development team are trained in the basic SPE techniques. In the early phases of a project, the developers can apply these techniques to construct simple models that support architectural and design decisions. This allows developers to get feedback on the performance characteristics of their architecture and design in a timely fashion. Later, as the models

become more complex, someone from the SPE Center can take them over to conduct more detailed studies that require more technical expertise.

The SPE Center develops tools, builds performance expertise, and assists developers with modeling problems. A member of this group may also review the team's models to confirm that nothing important has been overlooked. The central group can also develop reusable models or reference models, as well as provide data on the overhead for the organization's hardware/software platforms. Finally, the performance group can provide assistance in conducting measurements.

2.10 Ensure that Developers and Performance Specialists Have SPE Education, Training, and Tools

SPE consists of a comprehensive set of methods. Education and experience in these methods improves the architectures and designs created by developers. It helps performance specialists interface with developers, and shortens the time necessary for SPE studies. Performance tuning experience is helpful for SPE but it is not the same as proactive performance engineering. To be proficient you need additional education and training.

Tools are essential for SPE. Modeling tools expedite SPE studies and limit the mathematical background required for performance analysts to construct and solve the models. Measurement tools are vital for obtaining resource consumption data, evaluating performance against objectives, and verifying and validating results. However, simply acquiring a set of tools will not guarantee success. You must also have the expertise to know when and how to use them. It is also important to know when the result reported by a tool is unreasonable, so that problems with models or measurements can be detected and corrected.

The project team must have confidence in both the predictive capabilities of the models, and the analyst's skill in using them. Without this confidence, it is easier to attribute performance problems predicted by the models to modeling errors, rather than to actual problems with the software. If the developers understand the models and how they were created, they are more likely to have confidence in them.

2.11 Require Contractors To Use SPE On Your Products

You should require your contractors (e.g., external developers suppliers, etc.) to use SPE in developing your products to avoid unpleasant surprises when the products are delivered.

It is also important to specify deliverables that will allow you to assess whether SPE is being properly applied. These deliverables fall into four broad categories:

- *Plans*: These artifacts are targeted primarily at project management. They include technical plans for each development phase, as well as configuration management plans, policies, and procedures governing the production and maintenance of other SPE artifacts.
- *Performance objectives:* These artifacts include specifications for key performance scenarios, along with quantitative, measurable criteria for evaluating the performance of the system under development. They also include specifications for the execution environment(s) to be evaluated.
- *Performance models and results*: This category includes the performance models for key scenarios and operating environments, along with the model solutions for comparison to performance objectives.
- *Performance validation, verification, and measurement reports (V&V)*: This category includes documentation and measurement results that demonstrate that the models are truly representative of the software's performance, and that the software will meet performance requirements.

3.0 Performance Modeling Best Practices

These are best practices used by performance engineers who model the software architecture and design.

3.1 Use Performance Models To Evaluate Architecture And Design Alternatives Before Committing to Code

Today's software systems have stringent requirements for performance, availability, security and other quality attributes. In most cases, there are trade offs that must be made among these properties. For example, performance and security often conflict with one another.

It's unlikely that these trade offs will sort themselves out and ignoring them early in development process is a recipe for disaster. The "make it run, make it run right, make it run fast" approach is dangerous.

While it is possible to refactor code after it has been written to improve performance, refactoring is not free. It takes time and consumes resources. The more complex the refactoring, the more time and resources it requires. When performance problems arise, they are most often at the architecture or design level. Thus, refactoring to solve performance problems is likely to involve multiple components and their interfaces. The

result is that later refactoring efforts are likely to be large and very complex.

One company we worked with used a modeling study to estimate that refactoring their architecture would save approximately $2 million in hardware capacity. However, because the changes to the architecture were so extensive, they decided that it would be more economical to purchase the additional hardware. Another company used historical data to determine that its cost for refactoring to improve performance was approximately $850,000 annually [Williams, et al. 2002].

Simple performance models can provide the information needed to identify performance problems and evaluate architecture and design alternatives for correcting them. These models are inexpensive to construct and evaluate. They eliminate the need to implement the software and measure it before understanding its performance characteristics. And, they provide a quantitative basis for making trade-offs among quality attributes such as reliability, security, and performance.

3.2 Start With The Simplest Model That Identifies Problems With The System Architecture, Design, Or Implementation Plans Then Add Details As Your Knowledge Of The Software Increases

The early SPE models are easily constructed and solved to provide feedback on whether the proposed software is likely to meet performance goals. These simple models are sufficient to identify problems in the architecture or early design phases of the project. You can easily use them to evaluate many alternatives because they are easy to construct and evaluate. Later, as more details of the software are known, you can construct and solve more realistic (and complex) models.

Later in the development process. As the design and implementation proceed and more details are known, you expand the SPE models to include additional information in areas that are critical to performance.

3.3 Use Best- And Worst-Case Estimates Of Resource Requirements To Establish Bounds On Expected Performance And Manage Uncertainty In Estimates

SPE models rely upon estimates of resource requirements for the software execution. The precision of the model results depends on the quality of these estimates. Early in the software process, however, your knowledge of the details of the software is sketchy, and it is difficult to precisely estimate resource requirements. Because of this, SPE uses adaptive strategies, such as the best- and worst-case strategy.

For example, when there is high uncertainty about resource requirements, you use estimates of the upper and lower bounds of these quantities. Using these estimates, you produce predictions of the best-case and worst-case performance. If the predicted best-case performance is unsatisfactory, you look for feasible alternatives. If the worst-case prediction is satisfactory, you proceed to the next step of the development process with confidence. If the results are somewhere in between, the model analyses identify critical components whose resource estimates have the greatest effect, and you can focus on obtaining more precise data for them.

Best- and worst-case analysis identifies when performance is sensitive to the resource requirements of a few components, identifies those components, and permits assessment of the severity of problems as well as the likelihood that they will occur. When performance goals can never be met, best-and worst-case results also focus attention on potential design problems and solutions rather than on model assumptions. If you make all the best-case assumptions and the predicted performance is still not acceptable, it is hard to fault the assumptions.

3.4 Establish A Configuration Management Plan For Creating Baseline Performance Models and Keeping Them Synchronized With Changes To The Software

Many of the SPE artifacts evolve with the software. For example, performance scenarios and the models that represent them will be augmented as the design evolves. Managing changes to these SPE artifacts is similar to the configuration management used to manage changes to designs or code. Configuration management also makes it possible to ensure that a particular version of a performance model is accurately matched to the version of the design that it represents. While it isn't essential for many systems to have a formal configuration management plan, safety-critical systems and others require both the plan and the control of SPE artifacts.

Baselines for scenarios and models should be established following their initial validation and verification. Once an artifact has been baselined, it may only be changed using the established change control procedure.

The configuration management plan should specify how to identify an artifact (e.g., CustomerOrder software model v1.2), the criteria for establishing a baseline for an artifact, and the procedure to be used when making a change.

3.5 Use Performance Measurements to Gather Data For Constructing SPE Models and Validating Their Results

We have found that stakeholders are far more likely to believe initial model results when they are based on measurements of similar software or earlier versions of the modeled software. Even when you begin with estimates, use the models to study the sensitivity of model results to the estimates. Identify estimates with the greatest sensitivity, and obtain measurements to substitute for the estimates as early as possible.

Once parts of the system are implemented or prototyped, you can measure the resource usage of key components. Some portions of the system may be fully operational and can be measured, while other parts may still be in early or middle stages of design. Mix measurements of implemented components with walkthroughs of others to develop model specifications. Then you can more precisely evaluate the system as it evolves.

Measurements substantiate model predictions, and confirm that key performance factors have not been omitted from the models. Occasionally, software execution characteristics are omitted from a model because their effects are thought to be negligible. Later, you may discover that they in fact have a significant impact on performance. The way to detect these omissions is to measure critical components as early as possible and continue measuring them, to ensure that changes do not invalidate the models.

4.0 Performance Measurement Best Practices

These are practices for those responsible for measuring software performance and for performance testing.

4.1 Plan Measurement Experiments to Ensure That Results Are Both Representative And Reproducible

There are two key considerations in planning performance measurements: They must be representative and reproducible. To be representative, measurement results must accurately reflect each of the factors that affect performance: workload, software, and computer system environment. The goal is to design your measurement experiment in a way that balances the effort required to construct and execute the measurement experiment against the level of detail in the resultant data. When unimportant details are omitted, both the design effort and the overhead required to collect the data are reduced.

Reproducibility gives you confidence in the results. In order for a measurement to be reproducible, the workload, software, and computer system environment must be controlled so that you can repeat the measurement and get the same (or very similar) results each time.

The traditional scientific method is useful for designing and conducting performance measurements. The steps in the scientific method are:

1. Understand the purpose of the measurement—the questions to answer or the hypothesis to test.
2. Identify the data needed to answer those questions, along with the data collection tools and techniques to be used.
3. Identify experimental variables and controls. Ideally, if you are studying the effect of changing these variables, you only allow one variable to change at a time. You may be able to use multivariate techniques to analyze your data when this is not possible.
4. Define the test cases: workload, software, and the environment for each test.
5. Execute the tests and collect the data.
6. Analyze, interpret, and present the results.

All steps in this scientific method apply to SPE measurements. Experience shows that the following are vital to success:

- Careful design of the experimental test cases
- Identification of necessary data
- Collection and coordination of the measurement results

Defining the test cases means selecting the approach for representing the workload, software, and environment; and developing a test plan with priorities for the tests to be run.

Too much data obscures essential results, and may even perturb them because of the data collection overhead. Too little data reduces the experiment's usefulness, and may cause you to repeat an expensive set of measurements.

If the data comes from multiple sources, coordinate the measurements: Match the start and finish of the data collection intervals, and the granularity of the measurements.

For example, you cannot equate CPU utilization collected during an hour-long test of peak volumes with CPU utilization averaged over an eight-hour period.

It is unlikely that you will have time to run all of the measurements that you would like. Without a prioritized plan that defines the measurements to be performed,

you are likely to run out of time, only to discover that you are missing the most important data.

4.2 Instrument Software to Facilitate SPE Data Collection

You instrument software by inserting code (probes) at key points to measure pertinent execution characteristics. For example, you might insert code that records the time at the start and end of a business task to measure the end-to-end time for that task.

There are at least three reasons for supplementing the standard tools with instrumentation: convenience, data granularity, and control.

The first reason for instrumenting is to conveniently gather exactly the data you need. Although standard measurement tools may report SPE data, there are currently no tools that conveniently generate one report containing precisely the SPE data you need. Thus, getting the data is inconvenient at best. Some data, such as the sequence, frequency, and characteristics of user actions, is practically impossible to gather with standard measurement tools—you must analyze detailed traces and derive these events from them. Instrumenting allows you to tally the user requests within the software where they are easily identified, and produce convenient reports with exactly the data you need.

The second reason for instrumenting is that the data granularity from standard measurement tools seldom matches the SPE requirements. For example, suppose we want the end-to-end response time for a typical user session. Most data collection tools report performance data for online systems by "user interaction," that is, the starting event is the receipt of data or control information from the user, and the ending event is the transmission of a response. To use the standard measurement tools, you must gather data for each user interaction, and then calculate the session total from the individual times.

The third reason to instrument code is to control the measurement process. For SPE, we seldom need all of the measurement data all of the time; rather, we periodically need some of the data. Collecting data with standard measurement tools is not just a matter of flipping a switch; measurement requires many execution and data analysis steps. If measurements are infrequent, or if experienced personnel are unavailable, others must recreate the measurement steps. Instrumenting your code allows you to easily turn selected measurements on and off as needed.

4.3 Measure Critical Components Early and Often to Validate Models and Verify Their Predictions

Measurements substantiate model predictions, and confirm that key performance factors have not been omitted from the models. Occasionally, software execution characteristics are omitted from a model because their effects are thought to be negligible. Later, you may discover that they in fact have a significant impact on performance, as illustrated in the following anecdote:

An early life cycle model specified a transaction with five database "Selects." During detailed design, "Order by" clauses were added to three of the "Selects." The developers viewed the additional clause as "insignificant" because only one to five records would be sorted for each "Select." Upon investigation, though, the performance analyst discovered that over 50,000 instructions were executed for each sort!

The way to detect these omissions is to measure critical components as early as possible and continue measuring them, to ensure that changes do not invalidate the models.

5.0 Best Practice Techniques

These best practices identify techniques for working effectively with others including developers, management, and other divisions within the organization.

5.1 Quantify the Benefits of Tuning Versus Refactoring the Architecture or Design

Often tuning efforts that correct performance problems that arise due to a failure to employ SPE during software development masquerade as SPE successes. While tuning those systems can produce noticeable improvements, the resulting performance is unlikely to equal the performance of a well-architected system.

The best way to counter this difficulty is to expose tuning "success" for what it is—by comparing what was possible with tuning to what you could have achieved by using SPE effectively. For example, if a tuning project is constrained to use the existing architecture because of the extent of changes that would be required to modify the architecture, some simple SPE models could quantify the difference in performance achieved through tuning, versus the performance with an improved architecture. If possible, include an estimate of the total testing and development time required for the tuning approach versus the time to create models and build the software correctly from the outset. Collecting this data over time will provide a useful basis for constructing a business case for SPE for future projects [Williams and Smith 2003b].

5.2 Produce Timely Results for Performance Studies

Timely formulation and presentation of results and recommendations is vital, especially when corrective action is likely to be required. If a significant amount of time elapses between when the information is needed and when it is provided, key architectural or design decisions may have already been made. If this happens, the required changes may be more difficult to make, or they may no longer be feasible.

5.3 Produce Credible Model Results and Explain Them

The project team must have confidence in both the predictive capabilities of the models, and the analyst's skill in using them. Without this confidence, it is easier to attribute performance problems predicted by the models to modeling errors, rather than to actual problems with the software. If the developers understand the models and how they were created, they are more likely to have confidence in them. Be sure to explain how the models represent the software processing steps and interpret the model results. Demonstrating that models match measurements is another way to build confidence in the models.

You may need to establish the viability of the SPE models to predict the performance of your unique software problems. You will need to establish that the early models are capable of predicting the software's ultimate performance and identifying problems. Doing this when you adopt SPE promotes acceptance of the new technology and establishes the credibility of the analysts and the models.

5.4 Produce Quantitative Data for Thorough Evaluation of Alternatives

If you do uncover a problem, the bad news is more likely to be received well is you can also present alternatives for solving it. Quantitative data on the costs and benefits of alternatives will help the project team make the best choice from among the alternatives.

5.5 Secure Cooperation and Work to Achieve Performance Goals

User representatives, managers, designers, and performance analysts should form a cooperative team working toward the common goal of developing a product that satisfies quality of service objectives. The purpose of SPE is not to solve models, to point out flaws in either designs or models, or to make predictions–it is to make sure that performance requirements are correctly specified and that they are achieved in the final product.

6.0 Summary and Conclusions

This list characterizes the 24 best practices used by SPE practitioners. It should be clear from the list that proactive SPE means getting involved in new development projects early and working to build-in performance. For project management it identifies performance risk areas and applies effort commensurate with the risk. It tracks costs and benefits of SPE for future planning. It integrates SPE into the software development process and the project schedule. It establishes performance objectives and holds developers and managers accountable for meeting them. It focuses on critical parts of the system, and makes sure that the architecture can meet performance objectives for those parts. It uses quantitative performance models as the basis for architecture and design decisions. An organization that has a commitment to SPE at all levels; supports an SPE Center of Excellence; and has training, education and tools increases the likelihood of successful, timely deployment of high-performance systems.

For performance modeling, the software performance models provide the basis for evaluating architecture and design alternatives before committing to code. Best practices start with simple models, use best- and worst-case estimates of resource requirements to establish performance bounds and manage uncertainty, and match the model details to the software knowledge. They establish a plan for keeping baselined models that are synchronized with the software. They use measurements for model parameters and validation of results.

Performance measurement best practices plan experiments and ensure that results are representative and reproducible. They instrument software to facilitate SPE data collection, measure critical components early and often to validate models and verify their predictions.

Best-practice techniques for working effectively include collecting data to quantify the benefits of tuning versus architecture and design refactoring as a basis for constructing the SPE business plan. Modelers provide timely results, produce credible model results and explain them, and produce quantitative data sufficient for a thorough evaluation of alternatives. They secure cooperation and work to achieve performance objectives.

If you practice SPE now, this list will either give you confidence that you are on the right track, or give you some items for your "to do list." If you are new to SPE, get started and move in this direction as you get experience.

7.0 REFERENCES

[Boehm 1991] B. Boehm, “Software Risk Management: Principles and Practice,” I*EEE Software*, vol. 8, no. 1, pp. 32-41, 1991.

[Clements and Northrop 1996] P. C. Clements and L. M. Northrop, “Software Architecture: An Executive Overview,” Technical Report No. CMU/SEI-96-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[Javelin 2002] Javelin Technologies, “Best Practice Definition,” Oakville, Ontario, Canada, 2002 (www.javelin-tech.com)

[Reifer 2002] D. J. Reifer, Making the Software Business Case: Improvement by the Numbers, Boston, Addison-Wesley, 2002.

[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.

[Smith and Williams 2003a] C. U. Smith and L. G. Williams, “Ten Best Practices for Software Performance Engineering,” *MeasureIT,* Computer Measurement Group online Newsletter, June 2003.

[Williams and Smith 2002] “L. G. Williams and C. U. Smith, “PASA^{SM}: An Architectural Approach to Fixing Software Performance Problems,” *Proceedings of the CMG*, Reno, December 2002.

[Williams and Smith 2003b] “L. G. Williams and C. U. Smith, “Making the Business Case for Software Performance Engineering,” *Technical Report www.perfeng.com,* submitted for publication, June 2003.

[Williams, et al. 2002] L. G. Williams, C. U. Smith, Craig Hanson, Mary Hesselgrave, Thad Jennings, Panel: “The Economics of Software Performance Engineering,” CMG 2002, Reno, December, 2002.