Abstract

Software engineering has been defined as the procedures, methods and tools that control the software development process and provide the foundation for building high-quality software in a productive manner. There are many dimensions to "software quality", including, but not limited to, functionality, ease-of-use, flexibility, scalability, security and performance. Many of the software engineering methodologies focus on ensuring the software meets functional requirements while being produced within time and budget.

The performance requirement, however, is both becoming increasingly difficult to manage and more important to achieve. The move to client / server software architectures deployed on distributed systems has made traditional, queuing theory based analyses obsolete. This is compounded by Internet services in which information is sent to a user on a "subscription" rather than a "transaction" basis. Full-motion video, audio, and other multimedia support is expected from the underlying infrastructure as opposed to simple ASCII file transfers.

Just as the nature of the software and computer systems has changed, so has the user community. An enterprise no longer designs, develops and deploys systems for its own internal use; those systems are now intended for direct customer use. The growth of the Internet, particularly in the area of electronic commerce, has placed a higher emphasis on system performance.

This paper outlines a strategy by which the system's performance may be "engineered." That is to say, rather than developing the system and first identifying performance issues during system test, quality assurance or even initial deployment, a software manager can mitigate performance risk from the very onset of a system engineering project. Furthermore, this strategy also supports the need to determine system scalability, that is, given increased compute and communications resources, can the system support an increased workload.

Enterprise Information Technology and Performance

There can be little doubt that an enterprise's information systems – in-house applications, third-party applications, data, compute and communications systems – are of strategic importance. Whether or not those systems, and the management functions, have been "outsourced," the organization relies on these systems for virtually every facet of its business. If these systems don't function, the business doesn't function. If these systems cannot scale up, business growth is hindered. And if the systems perform poorly, the business performs poorly – loss in productivity, customers looking for alternatives, excess time and money spent in trying to fix a problem.

In many cases, system performance is evaluated far too late in the system life cycle. In the worst case, performance issues are discovered as the telephone rings; irate users or customers complaining about response time. In only a slightly better case, the organization may undertake benchmarking prior to deployment. In this scenario, however, the application has been designed, coded and tested leaving increased expenditures for compute and communication resources as the only recourse.

Ideally, performance engineering is integrated throughout the entire system engineering methodology. Most methodologies focus on the tools and processes to ensure functional correctness and to manage the development process. There is, however, little attention paid to the system's performance. One potential reason for this is that there are different groups within the Information Systems (IS) organization responsible for user applications, compute infrastructure and the communications networking. Too often, system performance is considered the domain of the "infrastructure" groups and not the application development staffs. Unfortunately, a poorly designed and developed application (from a performance standpoint) can become pathological. No amount of hardware or communications bandwidth can solve the problem.

The key question is: How can performance risk be managed and mitigated throughout the entire life cycle of the system?

The Short Answer

In a nutshell, performance modeling is used to support the system engineering project. Just as developers use data modeling, prototyping, and usage analysis to support engineering activities, performance modeling is focused on understanding and analyzing how the system will perform under a variety of circumstances. Performance modeling involves creating, validating and using a model of a system to produce estimates of the key performance metrics:

- Response time – how long will the system take to complete a particular piece of work such as a user transaction, a file transfer or some batch process.
- Throughput – how much work can the system support. Typical throughput metrics include transactions per second, maximum concurrent users and bytes per second.
- Utilization – what percentage of time are the system resources in use. If utilization is too high, there will be longer queuing delays, leading to higher response times. If utilization is too low, then perhaps too much money has been spent for excess capacity.

Ideally, system designers and developers will use all the tools and techniques available to mitigate performance risk. The use will not be haphazard, however. Activities supporting performance engineering will be integrated into the system development methodology. In doing so, the tools and techniques will be applied properly.

Basic Assumptions and Definitions

The Enterprise System

The overall enterprise system, for the purposes of this discussion, ranges from the business processes to the compute and communications infrastructure. The table below provides a basic delineation of the key enterprise system components, however it should be noted there are many areas of overlap:

Component	Description
Business Processes	The business processes drive the need for the system. As a user performs his or her work, the application(s) will be called upon to retrieve data, perform calculations and update information. Essential to performance engineering is an estimate of the volume of work to be performed.
User Applications	As mentioned, user applications support the execution of the business processes. When invoked, an application, either directly or via calls to other system services, use system resources. Key resources are: CPU Memory I/O Network User applications may be developed internally or purchased from third parties. Certain support applications, such as email or Intranet, are considered user applications in this document.
DBMS	The database management system (DBMS) provides facilities to organize and manage corporate information efficiently. In some cases, the DBMS has the greatest impact to an application's overall performance.

Middleware and System Utilities	This category covers a wide range of support software, including transaction monitors and systems management functions. These utilities provide services for the user applications and the general management of the system as a whole.
Communications	The communications infrastructure provides the connectivity between the various compute platforms in the system. Both the physical connectivity and the services offered by the protocols are included in this definition. The communications networks (LAN, WAN, etc.) provide the Network resource used by the applications.
Compute Platforms	The compute platforms execute the applications, DBMS's and other software necessary. These are typically distinguished into "client" or desktop machines and "servers." The compute platforms provide the CPU, memory and I/O resources.

As mentioned above, many IS departments are organized along technological lines. While each team can focus on the performance of the elements within its scope, true performance engineering takes into account the system as a whole.

Software Engineering Methods and Functions
Most IT development organizations use a certain methodology to develop applications. Although these methods vary, they typically include the following functions:

Function	**Description**
Requirements Analysis	Define the requirements – functional, reliability, cost, schedule, etc. – the system must satisfy.
System Architecture	Sometimes referred to the high-level design, the architecture specifies the "basic assumptions" to be used in constructing the system. These may include: Programming languages and tools Third party software, especially data base management systems (DBMS) Data distribution, that is centralized vs. decentralized data Supporting communications networks and technologies Compute platform operating systems and "middleware" Distribution of application functionality, client vs. server vs. multi-tiered services Required interfaces to other systems
System design	Given the general architecture, the system design produces detailed specification for the construction of application code.
Development	The code is developed in accordance with the design specifications.
Testing	The code is tested, generally at a variety of levels: unit test, sub-system tests, system tests, regression testing.
Quality Assurance	This is independent review of the system, including documentation, requirement audit trails and other management information, to ensure the system functions as required and will integrate into the existing infrastructure.

Deployment	The system is put into production. Deployment may be phased, that is the system is rolled out to groups of users as opposed to the entire user base at once.
Operation	Once the system is in production, there is typically an amount of maintenance and anticipated change required. Significant changes will cause the IT staff to initiate another project and iterate through the development life cycle.

Some notes and comments on the system life cycle:

- Again, the table above describes key functions that are performed within the context of a systems development methodology. There are many techniques to accomplish this work; furthermore the development methodology may be defined in either a linear ("waterfall") or iterative ("spiral") paradigm.
- Methodology discussions tend to view the compute and communications infrastructure as a given. Some attention is paid to the services provided by the infrastructure in the initial architecture and design functions. However, the software development methodologies tend to view the "hardware" as a separate domain from the software.

Benchmark
Many organizations have application and system benchmarking (model office, lab, testing, etc.) facilities. Separate from the production environment, these facilities allow the systems designers and developers to evaluate the system in a controlled environment. These facilities may have the ability to support performance benchmarking; specific testing to determine the performance of the system. In these cases, the following is assumed:

- Ability to generate a workload (user transactions, etc.) that represents the anticipated usage demand.
- Ability to measure and collect key performance metrics – response time, throughput and utilization.
- Ability to analyze collected data.

Metrics and ARM
The ultimate goal of any performance engineering initiative is to ensure the system is responsive, supports the workload and stays within budget. Systems provide measurements at various granularity to be used to help analyze system performance. Typical metrics include the following:

- Utilization – indicates what percentage of time a resource was in use over a given period
- CPU time – amount of time an application or process has spent using the CPU
- Memory – amount of memory used by an application or process
- I/O – amount of I/O generated, may also be a utilization of the I/O subsytem(s)
- Packet size and volume – Typically measured by a network device such as a sniffer or as part of a router's management information base (MIB), these statistics provide information regarding the network traffic.

Throughout this document, certain metrics are identified and discussed. For example, in the Requirements Analysis phase, an estimate of the amount of "think time" a user may take is used to determine total process response time requirements. When using modeling as described herein, these metrics are typically not described only by the average or mean value. The variation in the metric is usually incorporated, making for a more accurate and representative analysis. The analyst should be concerned with minimum and maximum values, the variation or standard deviation of the metric, and, if possible, and formal random number distribution that describes the metric.

Application Response Measurement (ARM) is an emerging set of standards and technologies (API's, collections agents, etc.) that allow the application developer to identify and collect response time measurements at various points in the application. The ARM technologies allow these metrics to be representative of the business process as a whole. For example, accepting a customer order may involve several application invocations to access the customer record, check credit, determine product availability and to issue picking and shipping directions. Using ARM, the application can report response times for each of these functions as well as the transaction as a whole. With this information, the analyst can separate "think time" from actual system processing time and determine which application functions contribute most heavily to any delay the user experiences.

Performance Engineering throughout the System Development Life Cycle

To achieve a responsive, scalable system, performance must be "engineered" throughout the life cycle. Trying to retrofit performance improvements will cost significantly more, both in terms of their implementation and the delay in deploying the system. The following sections take a step-by-step view in how to integrate performance engineering into the life cycle.

This view assumes the application is being developed from "scratch." However, much of this still applies when deploying third-party applications or modifying existing applications.

Requirements Analysis

Most of the effort in requirements analysis focuses on the functionality the system must provide. From a performance engineering perspective, key information must be obtained:

- Definition of the workflow or usage scenarios.
- Estimate the volume for each scenario.
- Determine response time requirements for the scenario as a whole and for critical intermediate points.

A usage scenario describes, at the business process level, the envisioned interaction between the user and the system. Typically, these scenarios make assumptions regarding the system's capabilities, architecture and design.

A usage scenario differs from a business process design. Business process designs, commonly performed using flowcharts, document not only the basic process flow but also the anticipated exceptions. A usage scenario is what the user is expected to encounter when executing the business process.

Scenario volumes are estimated based upon current and forecasted business activity. The intention is to ultimately determine the activity the system is expected to support during normal and peak periods. For each scenario identified, the following volume estimates are used:

- Total number of users who will perform the scenario
- Total scenario volume – for a given period (hour, day, week) how many times will this scenario be performed
- Scenario volume per user – how many times will a scenario be performed during a given period by a single user
- Maximum number of concurrent users, that is during the normal or peak periods, how many users will be performing this scenario at the same time.

It should be noted that the term "users" is used in the most liberal sense. Some activities use automatic scheduling of applications; in these cases a "virtual user" is defined. For example, a system may be configured to transmit performance and management information once every 15 minutes.

Response time requirements should be driven by the business needs. For each scenario, the following response times should be identified as targets:

- Total scenario duration, with an estimate of "think time."
- For each step, the amount of system "latency" or time allocated.

With this information, the scenario models can be developed. These models will be used to drive the application and system component models as they are developed.

System Architecture

While modeling system performance during the architectural phase(s) seems premature, it provides key decision making information:

- Help validate that the proposed architecture will support system performance and scalability requirements,
- Produce intermediate performance requirements to be used in system design and development activities,
- Provide direction regarding the deployment of systems management performance data collection activities,
- Begin to identify potential bottlenecks in the enterprise system.

The system architecture phase determines the basic assumptions to be used in designing, developing and deploying the application. One of two situations may exist: 1) the application will be deployed in an existing environment or 2) the entire environment is being developed from scratch.

In the first, an existing environment, work should be done to characterize it. This should be done "bottoms up," concentrating on the compute and communications resources. A baseline infrastructure model should contain the following:

- Network topology,
- Existing compute platforms,
- Representation of the current workload.

The representation of the current workload may be developed using current resource utilization metrics.

At this point, the baseline model should be executed and inspected to ensure it represents the existing system.

If an entirely new system is planned, the baseline infrastructure model will be developed using system proposal documentation, output from architecture activities and possibly system's developers best guesses. The baseline model represents the "initial stake in the ground" and will be modified as the system becomes more fully defined.

As the architectural activities progress, the following basic decisions are made and analyzed using the performance model(s):

- Placement of application functionality on the client and server compute platforms,
- Distribution or centralization of data,
- Use of middleware functionality,
- Basic information flows through the system.

Combining the baseline infrastructure model and the workload estimates with the architectural elements above, the analyst can develop a first cut model of the enterprise system. The applications will be "black boxes" from a modeling perspective. Key assumptions will be made regarding the resource usage of the application. These resource assumptions can be made by looking at analogous applications or experts' best guesses.

The analyst will use the model to explore various alternative architectures. Once the optimal overall architecture has been identified, the application resource usage assumptions become the intermediate level performance requirements (or “budget”) for the designers and developers.

System design
From a functionality standpoint, the system design activities take the architectural specifications and provide detailed system specifications. Similarly, the performance model(s) is refined based on the design decisions. As in the architectural phase, the model is used for the following:

- Validate system design decisions,
- Produce detailed performance requirements for development,
- Identify the most critical application processes (functions, DBMS calls, etc.),
- Provide guidance for the use of ARM technologies in development.

The “black boxes” used to analyze the architecture are replaced by the design information and estimates of their performance. The model is re-run and the basic performance of the system is assessed.

If the performance is lacking, the designers have several options:

- Refine the application’s resource usage estimates. This will result in more stringent development requirements.
- Increase the proposed compute power and / or network bandwidth. This will, of course, increase overall system costs.
- Review architectural decisions in light of the more detailed design.

By the end of the design phase, the performance model(s) should be of enough detail so that actual performance metrics may replace the assumptions

Development
Once development begins, the first “live” performance metrics can be taken. The assumptions made in the architecture and design phases are replace by actual application metrics:

- CPU usage,
- Memory usage,
- I/O requests,
- Network traffic.

If the actual performance metrics exceed the budget, the model may be used to examine various alternatives:

- “Stealing” time and resources from other parts of the application. Just as the resource requirements were under-estimated for this part of the application, there may other parts for which the requirements were over-estimated. By modeling higher performance (i.e. tougher performance requirements) in those sections, the analyst can determine that impact.
- Altering the design, either at the process level or for the system as a whole.
- Increasing system resources.

During the Development phase, the critical activity is to collect the performance information properly. At this point, the benchmarking or model office facilities begin to be used. If the application has been developed using ARM technologies, that information can be used as well. Key considerations in obtaining the performance information:

- Ensuring a controlled environment. The objective is to determine the performance of the application code “in a vacuum.” No other applications or unnecessary processes should be running that may skew the metrics, in particular response times.

- Database sizing. For applications that access databases, the test database should be sized appropriately. While modeling can help determine performance in the production data base environment, an extremely small or trivial database may produce performance metrics that are too small to measure.
- Understanding how the metrics are collected. When comparing metrics derived from actual observation with those from modeling, it is important to ensure an “apples to apples” comparison. For example, a model may “bundle” the operating system time required to load and schedule the application into the CPU and I/O requirements for that application. The metrics reported by the system may not account for this time as part of the application process.

Testing
System testing, particularly that done in a benchmarking environment, is used to validate functional correctness and produce performance information. For the purposes of this discussion, the Testing phase analyzes the system under an expected workload as opposed to the unit testing (done in development) that focuses on a small piece of code.

In general, the testing environment cannot recreate either the target environment or all the usage conditions for the application. The performance model, however, can analyze the system under all the expected conditions. In order to use the model in that fashion, it is important the model be validated.

During the Testing phase, the modeling and testing activities will mirror each other. The key goals are as follows:

- Determine system performance under conditions as close to expected operational conditions as possible
- Validate the performance model(s) to ensure that they are correct
- Identify any performance anomalies, that is unexpected performance issues or behavior.

In a benchmarking environment, there are three key concepts:

- System under test (SUT): the application, server(s), databases, etc. to be tested.
- System not under test (SNUT): the system used to drive the SUT. This system will typically collect response time metrics.
- Workload: the specific experiments - scripts and usage volume (number of users, frequency, etc.) – that will be used to analyze the application.

To validate the performance model(s), the benchmark environment and tests are modeled. The model’s performance information is compared against that collected in the benchmark. Key items to look at include:

- Application response times,
- Workload – did the model and the benchmark submit the same workload volume and, if controlled, at the same times,
- Throughput – did the model and the benchmark complete the same amount of transactions,
- Utilization – did the model and the benchmark show the same resource utilization,
- Other specific metrics – if available, other metrics such as queue lengths, may be used.

To produce the validation data, the model must closely match the benchmark environment as opposed to the expected operational environment. This may include:

- Changing the compute and communication models to that of the SUT.
- Modifying the workload representation and timings
- Ensuring the performance data reporting matches how the SNUT collected information.

If the model does not validate against the benchmark, the analyst can do several things:

- Examine both the model and the system closely to identify the source of variation, and if found, correct and rerun the validation experiments.
- Accommodate the variance by changing or "tuning" model parameters to match the observations. While this is somewhat risky, that risk can be mitigated by performing a sensitivity analysis. In this analysis, the overall impact to the system of each critical variable is determined. If the tuned variables are not particularly sensitive, then the risk is lessened.
- Accept the variation and understand the model's limitations in future analyses.

Quality Assurance
The Quality Assurance (QA) function is often assumed in the Testing phase. However, throughout the system life cycle, independent reviews of work product may be performed as part of an overall QA / risk mitigation effort.

Performance modeling is affected by the QA function in two ways: 1) as a supporting tool, and 2) as being subject to the QA process itself.

In its role as a supporting tool, performance modeling and analysis is used to demonstrate the system will meet performance requirements, thereby mitigating performance risk. Reviewing performance predictions in the Architecture and Design phases should be part of the overall review.

The validation efforts, particularly those during the Development and Testing phases, form the basis for assuring the model's quality. Throughout the modeling process (i.e. systems development life cycle) it is important to carefully document assumptions and sources of information.

Deployment
Once the system has passed testing, it may be deployed to the user community in the production environment. Deployment may be a complete roll-out or phased, depending on the size of the user community, the size, complexity and cost of the application, and the potential impact to the enterprise.

During the initial part of the Deployment, the following should occur as part of the performance engineering initiative:

- Validate the model using production metrics,
- Validate the workload estimate. In some cases, initial usage may be higher than anticipated as the users learn and experiment with the new system.
- To the extent possible, model and analyze the remainder of the system deployment. In a phased deployment, certain changes such as obtaining higher bandwidth circuits, more memory or faster CPUs may be feasible without jeopardizing budgets.

Operation
The last phase of a systems development initiative is to operate the system. This, however, does not mean the system remains constant. Typical changes that may affect system performance include the following:

- Increased system usage due either to an expanding user base or increased business activity,
- System changes to provide increased functionality
- Deployment of new applications on the shared infrastructure
- Deployment of new compute and / or communications technologies
- Deployment of new software and / or database management technologies, for example new versions or releases.

If the model(s) has been maintained and kept up to date, it can help analyze the impact of these changes and identify courses of action. To keep the model current, the following should be done:

- Maintain usage statistics to ensure workload volumes are accurate. The usage statistics can be analyzed statistically to determine growth patterns and forecast future demand for the system.

- Ensure any changes to the system are documented and tested to produce performance information. As changes are implemented, the old parameters should be replaced with the new information.
- Re-validate the model using current user and performance information periodically. This is probably best done when there are few “fires” or other major initiatives occurring.

Summary

This document has outlined an approach to integrating performance modeling and engineering into the systems development life cycle. By developing and using a performance model of the system as that system is developed, the system designer / developer can manage system performance to mitigate the risks of deploying a poorly performing system.