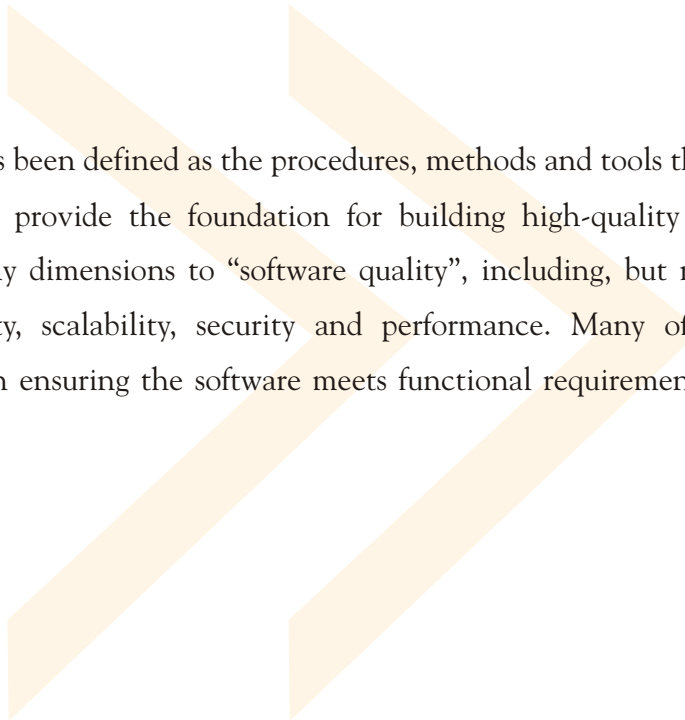


Mitigating Bottlenecks in Enterprise Solutions

White Paper - Version 1.0



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 analysis obsolete. This is compounded by Internet services in which information is sent to a user on a "subscription" rather than a "transaction" basis. Just as the nature of the software and computer systems has changed, so has the user community.

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.

1. Enterprise Technology and Performance: There can be little doubt that an enterprise's information systems – in-house applications, third-party applications, data-computation, 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 too and results in loss in productivity, customers looking for alternatives, excess time and money spent in trying to fix problems.

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 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 computation 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 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; performance bottlenecks and the communications networking. Too often, system performance is considered as the domain of the "infrastructure" groups and not the application development staffs. Unfortunately, a poorly tested application (from a performance standpoint) can become pathetic and contains a load of bottlenecks.

2 Performance Risk Management: 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 various circumstances and eliminating any performance bottlenecks. 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 the system can 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.

Early stages of Performance Validation

Performance: the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.

Validation: The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Validation of the responsiveness of systems includes: response time, throughput, and compliance with resource usage constraints. We consider the particular issues in the early testing stages and in pre-implementation stages. In pre-implementation stages complete validation is impossible because measurements of the final system are not yet available, factual information is limited and final software plans have not been formulated, actual resource usage can only be estimated, and workload characteristics must be anticipated. Extensive effort is required to study number of variations of operational scenarios possible in the final system.

The ultimate goal is to reduce performance failures (rather than guarantee that they will not occur). They increase the confidence in the feasibility of achieving performance objectives made in early life cycle stages. They provide the following information about the new system: Refinement and clarification of the performance requirements.

- Predictions of performance with precision matching the software knowledge available in the early development stage and the quality of resource usage estimates available at that time.
- Estimates of the sensitivity of the predictions to the accuracy of the resource usage estimates and workload intensity
- Understanding of the quantitative impact of testing alternatives, that is the effect of system changes on performance.
- Identification of critical parts of the testing process.
- Identification of assumptions that, if violated, could change the assessment.
- Assistance in designing performance tests

3. Validation for Responsiveness and Throughput

The general approach to early validation of performance testing is similar to any other testing evaluation process. It is based on evaluating a model of the design, and has five steps:

1. Capture performance requirements, and understand the system functions and rates of operation,
2. Understand the structure of the system and develop a model, which is a performance abstraction of the system.
3. Capture the resource requirements and insert them as model parameters
4. Solve the model and compare the results to the requirements.
5. Follow-up: interpret the predictions to suggest changes to aspects that fail to meet performance requirements.

4. Benchmark

The ability to support performance benchmarking is based on specific testing to determine the performance of the system.

- 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.

5. Metrics and ARM (Application Response Measurement)

The ultimate goal of any performance testing initiative is to ensure the system is responsive, Supports the workload and stays within budget. Systems provide measurements at various levels 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 subsystem(s)

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. 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.

6. Testing

System testing, particularly that done in a benchmarking environment is used to validate functional correctness and produce performance information. In general, the testing environment cannot recreate either the target environment or all the usage conditions for the application. 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
- Identify any performance anomalies that are 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.

7. 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:

- Supporting tool
- 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.

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.

About STC

STC ThirdEye Technology (India) Pvt Ltd is India's largest Independent software testing organization providing End-to-End testing Services. We build and operate dedicated India-based testing centers for our customers with the latest computing and data communication technologies, and deliver our services, with high standards of security and confidentiality. Consistent qualities of deliverables under compressed time schedules enable us to get repeat business.

We help Fortune 500 ERP, BFSI, Healthcare, Gaming and Telecom solution providers **We are ISO 9001:2000 certified organization.** For more details, please visit us at www.stcthirdeye.com

Disclaimer

The Whitepaper series presents reports on subjects in the sphere of activities of STC ThirdEye Technology (India) Pvt Ltd that are to be considered in the interest of wider public. These papers are part of the ongoing studies and authors will be glad to receive your comments.

The views expressed in these papers are to be regarded as those of the author and should not be interpreted as reflecting the views of the management of STC ThirdEye Technology (India) Pvt Ltd. STC ThirdEye Technology (India) Pvt Ltd assumes no responsibility for any actions taken by Anybody based on the information provided in this paper.