

## View Point



### Performance Engineering and Global Software Development

---

Sohel Aziz, Gaurav Caprihan, Kingshuk Dasgupta, and Stephen Lane

#### Abstract

The need to achieve system performance in a way that reduces risk and improves cost-effectiveness and time-to-market for performance-critical systems has become one of the principal goals of IS organizations. This requires a holistic and quantitative approach to performance, without compromising on other system qualities like flexibility, maintainability, reliability, and usability. Infosys has put into practice such an approach by combining recently developed practices in performance engineering with client delivery experience. Adapted to ensure collaboration among globally distributed application development teams and stakeholders, it combines workload and performance modeling with benchmarking, tuning, and optimization methodologies to deliver high-performance systems with reduced application development costs and lifecycles and improved productivity.

## Beyond “Build-Test-Tune”

Traditional approaches to system performance are inadequate in the complexities of today’s n-tier applications. Most of today’s enterprise systems can be characterized as “Build-Test-Tune”. This reactive approach typically addresses performance at the tail-end of the development process. Although it is possible to tune the platform or the code at this stage, it often yields only marginal benefits. Besides, it escalates development costs and delays schedules with time-consuming problem identification, analysis, and resolution activities. In addition, this reactive approach can clutter coding with optimizations, and thereby reduce maintainability in the long run.

The availability of cheap hardware platforms has enabled enterprises to deal with performance problems without formally addressing application performance. However, most of the benefits of faster and cheaper hardware have been offset by the multiplicity of operating systems, database management systems, application servers, etc. The “add more hardware” approach hides problems in the near-term, but they eventually show up in production.

## Proactive Enterprise Performance Engineering

Many projects have failed despite meeting functional requirements because their performance or other system qualities did not meet end-user expectations. This spawned extensive research and the development of a holistic view to system performance.

The result is a model-driven approach based on quantitative analysis and a thorough understanding of all the elements of system performance. The key dimensions of this approach are:

- System throughput requirements and response time – defined by business requirements and user expectations
- System parts that can be changed, tuned or optimized, including the architecture that interconnects different components, design of the components, configuration of platforms (e.g., amount of shared memory available for database server), and programs themselves
- All hardware and platforms (parts that can be bought) that cannot be modified. Builders must have a thorough understanding of the scaling characteristics of the hardware, i.e., whether they are scalable horizontally or vertically

The model-driven approach to system performance applies appropriate modeling, benchmarking, tuning, and optimization methodologies to ensure that all stakeholders are involved at the appropriate system requirement, design, development, and testing stages.

## Modeling

Models provide system designers with analytical and quantitative tools. The utility of Model-driven Architecture (MDA) has been proven in object-oriented design of systems. Applied to performance, models can be used to abstract various aspects of a problem. However, a model is useful only if it can provide reasonable accuracy and at the same time, is easy to build, use, and understand.

The extension of Unified Modeling Language (UML) sequence diagrams to “Software Execution Models” provides a preliminary understanding of system performance. Models provide a best-case analysis because they model the execution of a single instance of a scenario in isolation. This concept is well described in performance engineering research. There have been recent efforts to extend UML with stereotypes to annotate performance requirements.

Extension of “Software Execution Models” to accommodate concurrent execution of several instances of many scenarios requires Queuing Network Models (QNM). System behavior is commonly modeled by using queues to model the behavior of a single computing resource such as a CPU or a disk array and connecting such queues into logical networks (directed acyclic graphs) representing a deployment architecture .

Several standard QNM techniques have been extended. Layered Queuing Network (LQN) models are a popular extension of distributed and layered systems. Mean Value Analysis algorithms that provide average case solutions for QNM have shown reasonable results in several situations. Simulation techniques are also used to solve queuing models.

Workload models are extensively used to study system performance. They represent usage patterns of a system. System interactions or use cases are classified into workload classes based on similarity of performance characteristics, and their intensity is modeled as statistical distributions. Poisson distribution is typically used for interactive workload classes. Workload classes are derived from the use case model of the system or from a study of system logs and, in some cases, from time and motion study of system users.

Modeling obviates the need to host and maintain multiple test environments by using analytical or simulation techniques to extrapolate known observations and benchmarks. This enables assessment of a system's ability to sustain expected service level agreements. It helps identify indicative infrastructure configurations and also provides the ability to perform better impact analysis for application enhancements or infrastructure modifications.

### *Performance Testing and Benchmarking*

Another important element of the holistic approach to optimal system performance is reliable testing and benchmarking of systems. For this, workload classes must be simulated at various intensity levels, either in isolation or in combination with each other, to accurately measure response time and throughput.

Statistical validity of such benchmarks is of paramount importance – particularly to measure “service demands” that are used to parameterize queuing models. Load simulation tools such as Loadrunner from Mercury Interactive, Rational Robot from IBM, and QALoad from Compuware are quite popular. Open source simulators such as OpenSTA are fast gaining prominence.

However, it is not enough to measure response time and throughput during the testing and benchmarking process. It is important to monitor the hardware and the platforms through the entire course of testing and to correlate such measurements with the response time of each simulated transaction. Operating system monitors that are available with most versions of Unix and Linux are *vmstat* and *iostat*. *Perfmon* provides Microsoft sampled statistics for several hardware and operating system counters.

Some monitors are part of platform software such as databases or application servers, e.g. Oracle STATSPACK, SQL Server Performance Monitor, Websphere Resource Analyzer, etc. These tools record events and statistics that are relevant to the platform and are used for troubleshooting and tuning configurations and codes. Systems management software such as IBM Tivoli, HP Openview and BMC Patrol provide tools to gather data for monitoring.

### *Optimization and Tuning*

Performance engineering involves much more than tuning of badly performing software. System performance can be enhanced either by improving the “Architecture, Design, Code and Configuration” dimension or by acquiring more units of the “Operating Environment” dimension.

The architecture and design can be improved by leveraging the modeling techniques described earlier. Performance patterns and anti-patterns form a good basis for selection of favorable design choices. Platform-specific “best-practices” — for J2EE, Oracle, Rules Engines, etc — need to be assimilated by software organizations to leverage experience.

Finally, tuning of codes can be done with the help of developer support tools such as profilers and code-coverage analysis tools. Query profilers are commonly used for databases. It is best not to hack performance into systems because it can lead to compromises in the long-term maintainability of applications. Donald Knuth's advice in “Literate Programming” – “We should forget about small efficiencies about 97% of the time. Premature optimization is the root of all evil.” – cannot be overemphasized.

### *Capacity Planning*

The objective of capacity planning for applications and services is to ensure timely and efficient availability of sufficient hardware resources to satisfy user requirements. If IT managers focus on capacity planning, they can look ahead and minimize outages or poor performance due to unforeseen loads. It is not enough to build robust applications, their performance must be guaranteed.

In mainframe environments, capacity planning received a lot of attention because of the high cost of hardware. Conversely, there is a tendency to undermine the need for capacity planning for today's low-cost commodity hardware. The prevalent view is that there is little virtue in spending time and effort to build detailed models to predict required capacity when

computing platform is so inexpensive. Research shows that light-weight, “guerilla” capacity planning frameworks can be used to great effect on distributed systems. Detailed queuing theory- or simulation-based capacity planning models can be used for large-scale deployments.

### Performance-Centric Development

As noted above, a structured set of activities performed through the development lifecycle is one approach to address system qualities. These activities can also be performed in a waterfall model and, with some minor modifications, as parts of iterations, if iterative development methodologies are followed.

A performance architect is required to oversee, own and deliver all the activities. Performance requirements and service level agreements (SLA) must be negotiated with business stakeholders. The performance architect must negotiate with system architects and designers on the architectural choices to adequately address performance needs. The performance architect must also work with the development team, test team, and system administrators to test and benchmark the system and plan capacity to ensure efficient deployment (Figure 1).

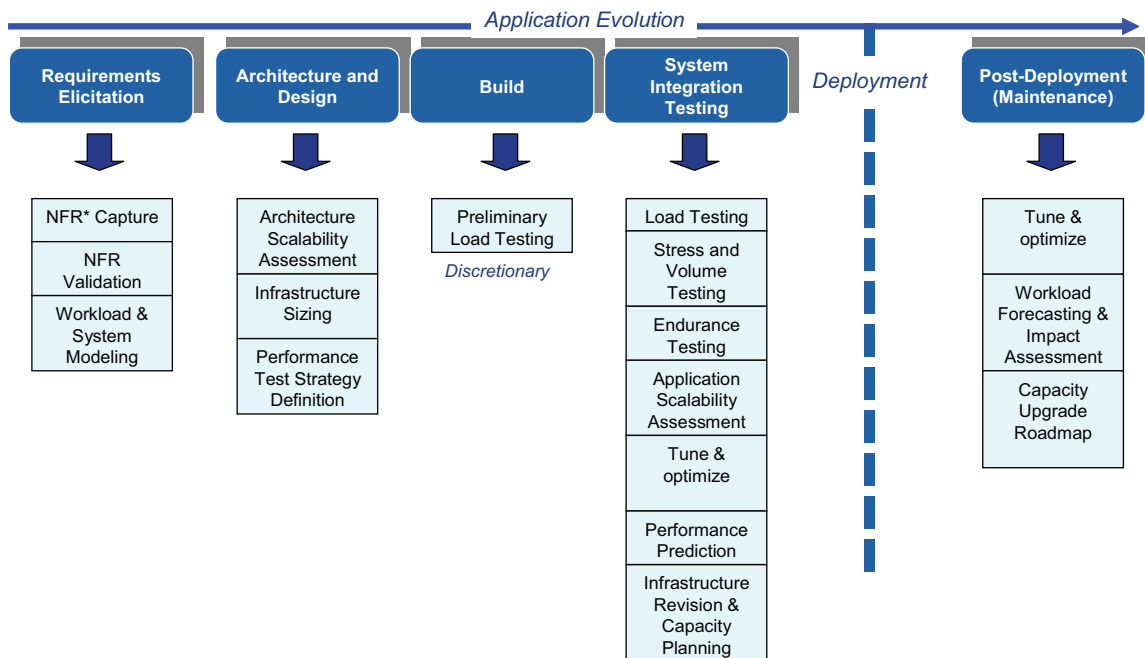


Figure 1: Performance-Centric Development Model

### Organizing for Performance

The key to performance-centric development model is collaboration. This is particularly true when the stakeholders, architects, developers, and other participants in the process operate in a distributed fashion, for example the global delivery model of software development. One of the key benefits of this model is its ability to leverage optimal skills and location for any activity in the lifecycle. Geographical and logical separation of activities that have traditionally been kept together is possible because of the emphasis on communication and processes.

Activities that require intense interactions with key stakeholders, are contextua or have little benefit from scale are best done in close collaboration between the partnering organizations. Other activities can leverage scale to lower the cost of offshore operations. Figure 2 illustrates a selection of activities in a globally distributed development model.

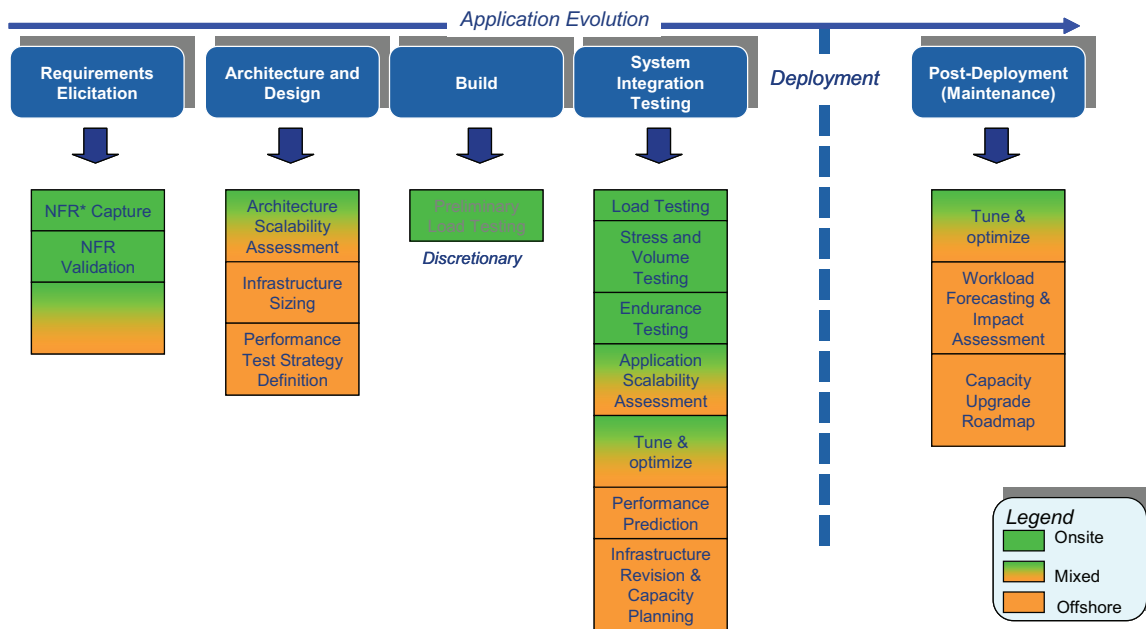


Figure 2: Collaboration for Performance Engineering in a Global Development Model

Figure 2 shows the distribution of activities between onsite and offshore for an organization which does testing onsite – on the target deployment infrastructure. Often, applications are tested for load and stress in offshore testing labs.

## Conclusion

System quality attributes and performance are as important as functional correctness and completeness. However, performance is usually “tuned” into applications as an afterthought and is often inadequately addressed in the development lifecycle process.

Performance engineering calls for specific skills in model building, monitoring, benchmarking, optimization and tuning. Several activities have been identified through the development lifecycle and beyond to meet the skill needs. These activities must be owned by a performance architect who works with different members of the development organization.

The widespread adoption and maturity of global software development has provided access to skills in model building and also monitoring, benchmarking, optimization and tuning. However, a collaborative framework is needed to carry out performance engineering activities in an efficient and cost-effective way.

Models can be applied not only to internally distributed development organizations but also to third-party sourcing relationships. This will enable companies to take advantage of a worldwide pool of highly trained specialists representing different disciplines, while maintaining essential activities and knowledge in-house.

## About the Authors

**Sohel Aziz** is a Principal Architect and EMEA Practice Lead for Technology Consulting at Infosys. He has over 13 years of experience as technology architect, technology programme manager and analyst. He has extensive experience in Enterprise Architecture assessment and definition, technology product selection, and technology business case development. His current areas of focus are Enterprise Architecture and effective governance models for ensuring business-IT alignment by leveraging architecture.

**Gaurav Caprihan** is a Principal Architect at Infosys. He heads the Quality of Service (QoS) team at the Software Engineering and Technology Labs (SETLabs), Infosys' research group. He has over 12 years of experience in applied research in the industry. His current research is focused on managing the performance of software applications as they evolve under the globally distributed software development paradigm.

**Kingshuk Dasgupta**, contributed to this white paper during his tenure as a Principal Architect with Infosys. Kingshuk has over 12 years of industry experience in designing, implementing and managing the development of software solutions. While at Infosys, he specialized in Performance and Reliability Engineering, Object-Oriented Analysis, Design and Programming, and Databases.

**Stephen Lane** is a Senior Manager - Marketing at Infosys. In his current responsibility, he works with a number of Infosys' leading clients to identify, document, and disseminate global sourcing best practices. In 15 of his 27 years in the IT industry, he focused on outsourcing. In that time, he served as a project delivery manager on the vendor side and as an outsourcing consultant to financial services companies. Prior to joining Infosys in 2004, Stephen was a global sourcing research analyst, writer, and speaker in the U.S., Europe, and Asia



For more information, contact [askus@infosys.com](mailto:askus@infosys.com)

### About Infosys

Many of the world's most successful organizations rely on Infosys to deliver measurable business value. Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 30 countries build tomorrow's enterprise.

For more information about Infosys (NASDAQ:INFY), visit [www.infosys.com](http://www.infosys.com).