IBM Global Services

From Realism to Reality: Making the Case for Performance Engineering

Presented at the Workshop on Performance and Reliability
October 2nd – 4th, 2003
New York, NY

Dave Jewell
Consulting IT Specialist, IBM Corporation
455 Park Place
Lexington, KY 40511 USA
e-mail: jewell@us.ibm.com
Phone: 1-859-243-3338

Introduction

When done well, performance testing can bring needed discipline and accountability to bear in the development of IT solutions. However, even a realistic performance test can be "too little, too late", especially in a project where performance problems are rooted in architectural and design choices that cannot be corrected by mere coding fixes. In this paper, the author recounts his personal experience with one such project and gives examples of how the proper use of performance engineering techniques can help architects, developers and testers get "reality checks" on system performance earlier in the development process.

Note: The observations, viewpoints and conclusions expressed in this paper are those of the author, and do not necessarily represent the viewpoints of IBM or its management.

About the Author

Dave Jewell is a Consulting IT Specialist with IBM Global Services. In his 25-year career with IBM, he has held a variety of technical positions related to software development and testing. Since 1989, Dave's assignments have mostly related to performance testing, performance analysis or performance engineering. More recently, Dave served as the lead subject matter expert in the development of two online performance testing classes that are now offered to IBM employees via IBM's internal education directory.

Dave and his family reside in Lexington, KY.

1 Setting the Stage

1.1 The Story Begins

The story began in early 1998, when an IBM Global Services project team was assembled to re-engineer an IBM internal service call management solution that had been used successfully in North America, as a first step toward deploying the re-engineered solution worldwide. The main components of the existing solution which were to be changed are generically described in the following table.

Component	Description	Platform (OS / Middleware / Database)	History	Networking Protocol
Voice call management / Core repository	Traditional centralized call center representatives accept voice calls for service and enter them in system	MVS[1] / CICS[2] / IMS[3]	Developed in early1980s, regularly enhanced	SNA[4] LU2 (mainframe terminal session)
Electronic call facility	Accepts service call data from customer-managed call centers, provides call status updates	MVS / CICS / DB2[5]	Developed in early 1990s, regularly enhanced	SNA LU0, SNA LU6.2, TCP/IP
Technician communication facility	Exchanges service call status and dispatching information with field technicians' hand-held devices	Proprietary network hardware and software technology bridging between mainframe and private RF network environments.	Developed in mid-1980s, periodically enhanced	SNA LU6.2 (interface between bridge and mainframe)

[1] Multiple Virtual Systems (MVS) was and is an IBM mainframe operating system which is now encompassed in the IBM OS/390 product family.

[2] Customer Information Control System (CICS) was and is a family of IBM online transaction processing (OLTP) middleware products.

[3] Information Management System (IMS) was and is an IBM mainframe product which included both hierarchical database and OLTP functionality. In this project, only the hierarchical database feature of IMS was used.

[4] Systems Network Architecture (SNA) was and is a proprietary IBM network architecture. The applications described here used LU0, LU2 and LU6.2, which are specific SNA communications protocols.

[5] DB2 was and is a family of IBM relational database products.

1.2 Good Intentions

Key requirements and implementation features of the project included the following:

1. The addition of a scheduling component to further optimize and automate dispatching of the field technicians.

2. The enhancement of the core repository to support worldwide deployment, especially with respect to non-North American geographies.

3. The conversion of the core service call repository database from IMS to DB2.

Due to the perceived business urgency of this solution, the project plan for the first release of the re-engineering project had an aggressive schedule with high risk, factors which would come to haunt the project later. Work began on planning for the performance testing in July 1998. The performance test plan had the following major characteristics.

- Because the existing components had all undergone performance testing before, the performance test effort would benefit from the ability reuse most of the automated performance testing scripts which would be needed.

- IBM Teleprocessing Network Simulator (TPNS)[6] was used for the performance testing of all components except one. For the lone exception, the technician communication facility, the "bridge" workload was simulated using custom C++ tools running on an AIX / RS/6000 server.

- While the coordination of all of this automated testing was a complex undertaking, the basic approach to the test was fairly simple; simulate the projected load for a peak hour for both the existing system and the new system, then compare the results of the two tests.

- Since the major mainframe test systems shared a large partitioned mainframe with the production system, most major performance tests had to be scheduled off-shift to minimize interference between production and performance test.

- One of the lead architects for re-engineered solution was a big believer in the value of performance testing, and was personally on hand during many of the off-shift tests to monitor the performance of the systems under test.

Systemic performance problems led to the cancellation of the project after over 18 months and $3.5 million had been spent. The reasons for this contributed to the author's decision to become an advocate and practitioner of performance engineering.

[6] Teleprocessing Network Simulator (TPNS) was and is an IBM network simulation product used for a variety of purposes, including load/stress testing, regression testing and validation of proposed network environments. Its primary strength is its support for SNA and other IBM legacy networking technologies.

2 So What Happened?

2.1 Read 'em and Weep!

As one might expect, the project encountered numerous problems in addition to those related to performance. The project started to run behind schedule and over budget. Numerous functional defects were found, and the initial results of performance tests of the re-engineered system were cause for concern, especially with respect to the core service call management repository. However, at the time these were not viewed as insurmountable problems. Fixes were delivered for many of the functional defects, and extra time was given to revisit some of the database-intensive programs, with the rationale being that many of the needed improvements would be gained by applying standard relational database performance improvement measures (e.g. eliminating table scans and index scans).

For the functional defects, this optimistic outlook appeared to be justified. Many defects were resolved and overall defect rates declined, indicating that the re-engineered solution was being to stabilize functionally. However, performance defects, although few in number, were *not* being resolved. Even after additional weeks were spent attempting to improve performance, one test showed that the new solution used *five times* as much CPU as the current solution for the core service call management repository. While the cause was not fully understood, suspicion now turned to newly developed database access routines being used to manage access the DB2 table.

By now it was April of 1999, and it was apparent that these problems could not be resolved prior to IBM's self-imposed Y2K "freeze" deadline. Furthermore, the magnitude of the excessive CPU utilization made it infeasible to resolve the problem simply by allocating additional mainframe capacity. As a result, the deployment of this release was canceled. While some solution components were used in later releases, the customer organization elected not to attempt to deploy the re-engineered solution as a whole in 2000 after the Y2K freeze had been lifted.

2.2 What Can We Learn?

It is easy to be critical of the handling of this or any other failed project, especially having the luxury of hindsight. However, the reason for citing this example is not to criticize, but to glean the lessons learned so that hindsight from this project can become foresight which can be applied to future projects.

From a testing perspective, this project served as an object lesson in the difference between functional defects and performance defects. Experienced performance testers understand that their job is in many respects different than that of functional testers. Though both may use automation, the tools and their usage are often different, the means of detecting non-conformance is different, the nature of the non-conformances found is very different, and very often the level of effort required to resolve problems is different. In this case, functional quality (as measured by defect trends) eventually improved as functional testing and remediation ran its course, whereas performance problems tended to be stubborn and unyielding. This would indicate that defect rate models (used to predict when a release will be ready to ship based on statistical defect trends) may not always be as helpful in dealing with performance readiness as they might be in dealing with functional readiness.

From a development perspective, the challenges faced when significantly modifying a mature system were underscored. In the many years preceding this re-engineering project, a great deal of work had already gone into getting the most out of the CICS (OLTP) and IMS (database) technologies used in the legacy solution. Furthermore, the bulk of the application programming had been done in an IBM internal language known as PL/S, which allowed nearly the same execution efficiency as assembler language programming. The re-engineered solution replaced IMS with DB2 (for which some performance impact was expected) and introduced C++ object-oriented database access routines to enhance portability. It was the latter implementation choice to which the drastic increases in CPU utilization were ultimately attributed.

From a project perspective, the performance risks discussed above were not brought to light in time to identify and resolve impacts before deployment. The pressure to deliver a solution quickly would have discouraged the development team from spending time on early performance studies to validate the design approach, even if they had anticipated the risks in this area. Although the performance team succeeded in delivering a "realistic" performance test, the "reality check" came too late in the process to facilitate a successful deployment.

The bottom line: We *must* find better ways to identify and mitigate performance risks by introducing performance "realism" into earlier parts of the development cycle.

3 Addressing Performance from the Perspective of Risk

In many respect, IT performance specialists have a great deal in common with our colleagues in project management. Both disciplines deal with the trade-offs between resource, work and time, and both disciplines must manage risk. The follwing list characterizes approaches that might be used to managing increasing levels performance risk.

- **Performance verification** - Regression test of a system with minor changes, with little to no change in performance attributes expected
- **Performance validation** - Testing to performance requirements where performance risks are "moderate", assuming development has used reviews and testing to address performance concerns as well
- **Performance engineering** - Performance risk is centrally managed throughout the development / test / deployment cycle, especially for complex solutions where performance risk is high. A variety of techniques may be used (modeling, prototyping, testing, monitoring) to achieve this end.

It can be argued that all of these approaches are in fact different levels of what we refer to as performance engineering, with the level of effort and scrutiny made commensurate to the level of performance risk. To use a medical illustration, proper risk management for our re-engineering project example could be compared to risk management used for a heart transplant.

Area of Comparison	Re-engineering Project	Heart Transplant
Objective	A change to the core database is needed to extend an application's useful life and scope.	A heart transplant is needed to extend the length and quality of the patient's life.
Managing Quality Risks	Reviews and testing will be done to make sure the new application works as designed.	Careful planning, training, rehearsals and monitoring take place to make sure the operation is done correctly and proceeds smoothly.
Managing Performance Risks	Prototype testing of the database access routine takes place BEFORE the overall performance test to make sure that the re-engineered solution can perform successfully.	Careful testing and examination of the real (or artificial) heart to be transplanted is done before the actualtransplant operation takes place.

Clearly, then, there are times that additional emphasis must be placed on using methods (such as the early performance prototyping suggested above) that will give us earlier answers to our performance questions. With this in mind, IBM views performance engineering (or PE) as "a technical discipline which aims to ensure that a development project results in the delivery of a system which meets a pre-specified set of performance objectives". In the words of one of IBM's UK PE practitioners, we want to ensure performance is never a "nasty surprise" encountered at the end of the project.

4 Making Performance "Reality Checks" Happen Earlier

In light of what has been discussed, performance testers face something of a dilemma. In spite of the clear need on the part of solution architects, designers and developers for earlier performance feedback, performance testers using traditional methods cannot necessarily oblige. It may take weeks or months to assemble the infrastructure, populate the production size databases and create the automated load testing scripts and procedures needed to conduct a thorough performance test; even then, such a test is generally dependent on the completion of application development.

Performance engineering introduces techniques that enable performance "reality checks" to take place throughout the software development lifecycle. Some examples of these techniques are listed below. Note that many of these techniques provide opportunities for early cooperation and synergy between development and test groups.

- **Performance testing during unit test** – Developers may wish to consider including some kind of performance testing as part of their unit test activities. For example, timings could be collected for critical transactions, database queries could be analyzed for improvement and so on. It may be appropriate for some of the unit test to take place in the performance test environment, since larger databases may give a better idea of how performance will be in the production environment, and the performance test environment may contain larger databases than are normally available to developers.

- **Prototyping** – Performance prototyping as a technique has already been mentioned. In many cases there will be more than one technical alternative under consideration by software architects and designers, with each alternative having different performance attributes. A performance prototype can be thought of as a partially developed application, with enough logic in place to allow assessment of one or more technical alternatives. For example, in one IBM project a prototype was used to mimic future database access activity against each of three different data mirroring solutions to assess their respective effects on transaction throughput.

- **Modeling** – Performance modeling techniques allow performance attributes of a proposed system to be estimated or projected as a means of understanding that system's service level characteristics. Models may be developed using something as simple as spreadsheets, or they may require the use of complex modeling tools. While such models are usually the responsibility of a performance engineer or architect to develop, modeling provides an opportunity for great synergy with the performance testing team. Performance test results can be used to calibrate and improve the model, while modeling results may point to areas where more in-depth performance testing may be required. In many cases, modeling can be used to amplify or extend the results of performance testing by running modeling scenarios that cannot presently be tested (e.g. changing hardware or capacity assumptions, increasing volumes, etc.).

- **Requirements** – Very often the performance aspect of system requirements does not get enough attention. Performance requirements may not be defined at all, or they may be vague or incomplete, making it difficult to determine whether the requirements have truly been met and adding to the risk of failure. Both the performance engineer and the performance tester have a vested interest in seeing that the performance requirements are fully elaborated from both a business and technical perspective, so that they meet the so-called "S.M.A.R.T." criteria (Specific, Measurable, Attainable, Realistic, Testable).

5 Conclusion

Managing performance risk effectively means that we must be willing to give the appropriate level of attention to performance at the most opportune times during the development cycle. This does not diminish the importance of performance testing, which remains a critical assessment task. Rather, this means we must begin to combine performance engineering and performance testing techniques if we are to be successful in deploying tomorrow's increasingly complex, critical and interdependent systems. To that end, the following actions are advocated to bring about the "culture change" needed in our respective organizations.

- Be an advocate for early and aggressive management of performance risk throughout the software development lifecycle.
- Find ways to get involved earlier in the software development lifecycle than you are now.
- Become the "performance conscience" for your organization or area.
- Push for multidisciplinary performance engineering teams that have the right mix of skills to assess and address performance risk.