White Paper

Performance Testing Methodology

by Johann du Plessis

Introduction

One of the main concerns with a performance testing project is how much value the testing adds. Is performance testing worth the investment? To be sure about this, the standard of work delivered must be measurable.

To start this process I reviewed some of the projects I've been involved in over the past few years. I had a good look at successes and what was done on the projects that were the most successful. I was surprised to find that without knowing it, I followed the same methodology on all these projects.

Adding Value

How do we determine (measure) the value we are adding to a performance testing project or whether any value was added at all? Was the testing successful? Did the effort highlight problem areas?

A successful performance testing project will include all or most of the following:

- **Successful tests**
  What is a successful performance test? This can be viewed in more than one way. The test can be successful because all requirements were met with no errors to report. However, if the test fails because of a performance or load related problem, it is still a successful performance test because the problem was identified.
- **Problems identified early**
  Finding defects early saves money. This is just as true for performance testing as it is for functional testing. Performance testing should be started as early as possible to find problems before they are difficult and expensive to identify and fix.

  "Finding defects early saves money"
- **Improvement in system performance**
  Improvement in system performance is one of the main reasons the testing is done in the first place and the value added by this is huge.
- **Non-performance problems identified**
  This includes defects that would not be detected easily during functional testing. A good example of this is a counter of the number of transactions for a day. I recently tested an application where the requirement was 9999 transactions per day. This was a counter in the database that was not displayed in the application. At some point my load test users started failing. Resources and performance were fine, but I passed 999 transactions for the day and the application only catered for that, not 9999. No defect was raised because functional testing never reached 999 transactions in a day.
- **Visible deliverables**
  It's great to talk about what you found, but showing it off (making it visible and available) is even better. Physical deliverables make it easy to communicate successes and show how time was spent.

  The methodology described in this paper has the following physical deliverables:

  - Assessment report
  - Test strategy/plan
  - Test scripts
  - Test scenarios
  - Test results

- Results summary(s)
- Test report and presentation

Methodology

"Following a proper methodology guarantees a successful performance test project"

Following a proper methodology guarantees a successful performance test project. It is important that the same methodology is followed on all projects. This ensures that the same deliverables are produced and quality can be measured against this. Comparisons can be made between different projects by evaluating the deliverables. Projects that were the most successful are identified easily and it is also easy to identify what went wrong or was done wrong on the projects that were less successful or where less value was added.

Value added can be determined by looking at system improvement as plotted on graphs or given as percentages for areas where improvements were made. In *Figure 1* two average response time graphs are shown. The graph on the left shows the response time for the first test and the graph on the right for the final test of the performance test project and the improvement (value added) is obvious. These graphs are available after testing so show it to the people that matter.

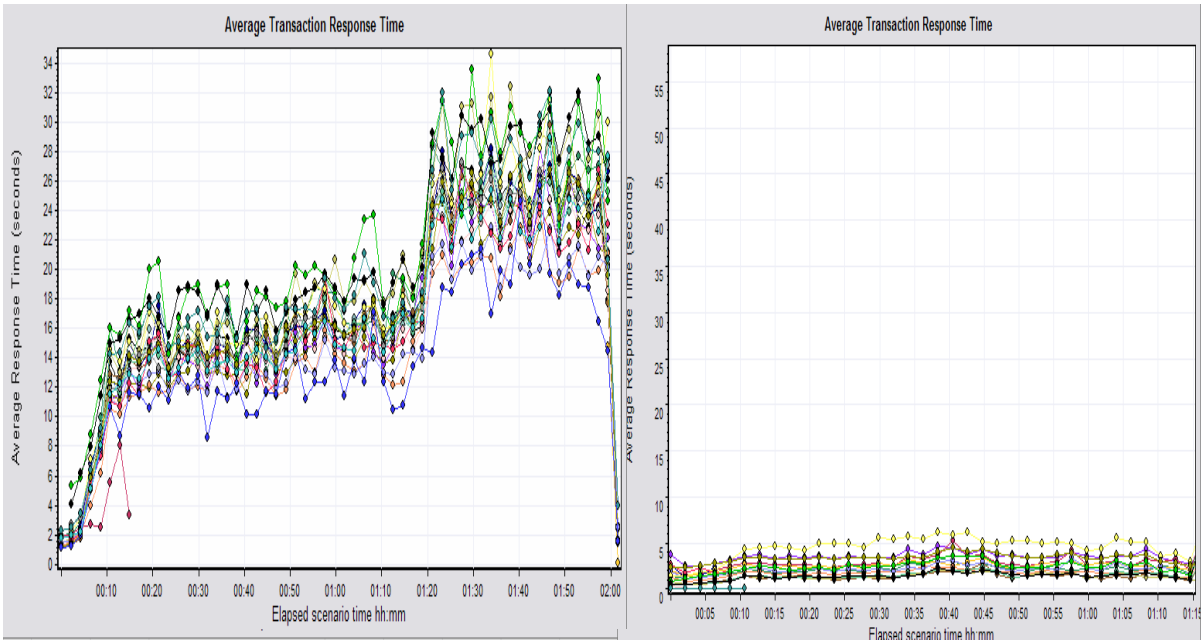

Figure 1 - First and final test results compared

The performance testing methodology consists of six phases. Each phase is completed with a deliverable or deliverables. *Table 1* shows each phase with the expected deliverable(s).

Phase	Deliverable
1 - Project assessment	Assessment report
2 - Planning	Test plan
3 - Scripting	Test scripts
4 - Test execution	Test scenarios, test results
5 - Results analysis	Results summary
6 - Reporting	Performance test report and presentation

Table 1

Phase 1 - Project Assessment

The most successful projects that I have completed all started with a formal assessment. This phase determines whether the work can be done and if so, how? The assessment is a process of gathering information. Requirements are analysed and the system and architecture studied to determine whether the requirements can be met with what is available in the specific environment. People should also be informed about your own requirements to perform the testing, including the time required to get useful results.

"Expectations should be managed for the duration of the project"

Expectations should be managed for the duration of the project. People usually assume that results can be delivered very easily and very quickly. If you are using some automation tool, as is usually the case with performance testing, this is even more applicable. The assessment process is where expectation management starts.

Requirements

The requirements for performance testing are usually very specific. When requirements are not known at the start of the assessment, finding out what they are should be part of the assessment process. Gather as much information as possible on every detail. This is crucial to the success of the project. All the information will be used to produce good deliverables. Where something is not possible, communicate this properly to help manage expectations. In *Table 2* we look at some of the key areas the assessment covers.

Project Assessment	
What must be achieved? (Business problem to solve)	• Number of users • Acceptable response times • Business processes to test • Baselines • Data volumes
Architecture / Platform	• Are you familiar with the architecture? • Do you have experience with the architecture? • System components (Hardware & Software)
Test environment	• Suitable for performance testing? • Hardware • Software
Which tool will be used?	• Are you familiar with the tool? • Is the tool compatible with the architecture? • Hardware & software requirements for tool installation and use
Monitoring	• What must be monitored – requirements? • What can be monitored? • Requirements to put monitoring in place
Available time	• Time available vs. time required • Give yourself enough time • Manage expectations • Enough time = meaningful results
Requirements to perform testing	• Access to key people • Hardware requirements • Software requirements • Data requirements

Client expectations	▪ Don't say yes to everything ▪ Point out limitations on your side ▪ Highlight possible risks ▪ Highlight any exclusions from the testing and the reasons for excluding the area(s)
Assessment report	▪ Can the work be done? ▪ How will the work be done? ▪ Who will do the work ▪ Time and effort required ▪ Exclusions ▪ Deliverables

Table 2

A report communicating all the findings is drafted at the end of the assessment. It includes decisions made on how to proceed with the project and the estimated time needed to complete the testing. The table of contents of a typical assessment report is shown in *Figure 2*.

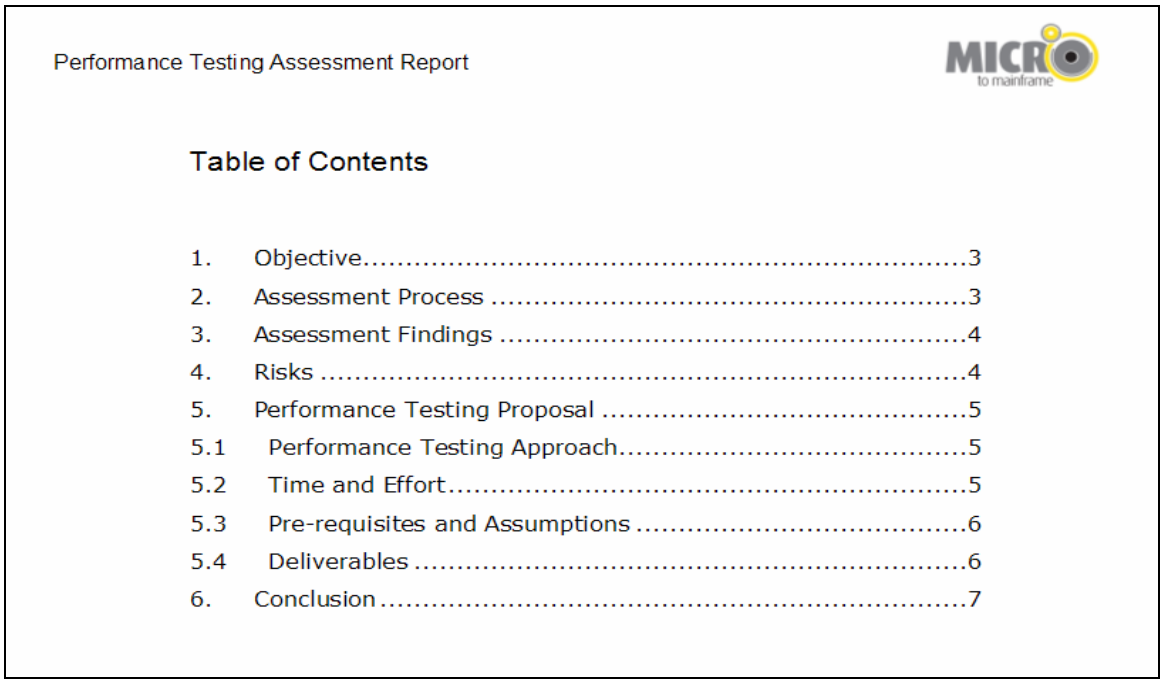
Performance Testing Assessment Report

Table of Contents

Figure 2

Phase 2 - Planning

The information gathered during the project assessment is used for planning the performance testing and to start the test plan. The performance test plan must contain all the detail and acts as a check list and reference for test execution. The test plan forms the backbone of the testing process and is a working document that is updated as the project progresses. A fully completed test plan guarantees that no details are left out for the test execution.

"A fully completed test plan guarantees that no details are left out"

Please note that this paper does not cover the content of the test plan. However, there are a few **key items** that should always be included in the test plan. These are:

- Goal
- Objectives
- Scope
- System diagram
- Exclusions
- Monitoring
- Responsible people
- Environment
- Test hardware requirements
- Test software requirements
- Test data requirements
- Test tool requirements
- Security access
- Test scenarios
- Test execution
- Results analysis
- Report and feedback

Completing the test plan is crucial as it requires the completion of the sections that cover the requirements for testing. This includes resources and other logistics that need to be in place before successful performance testing can start. A completed test plan gives the assurance that everything needed is or will be in place for test execution. The table of contents of a typical performance test plan is shown in *Figure 3*.

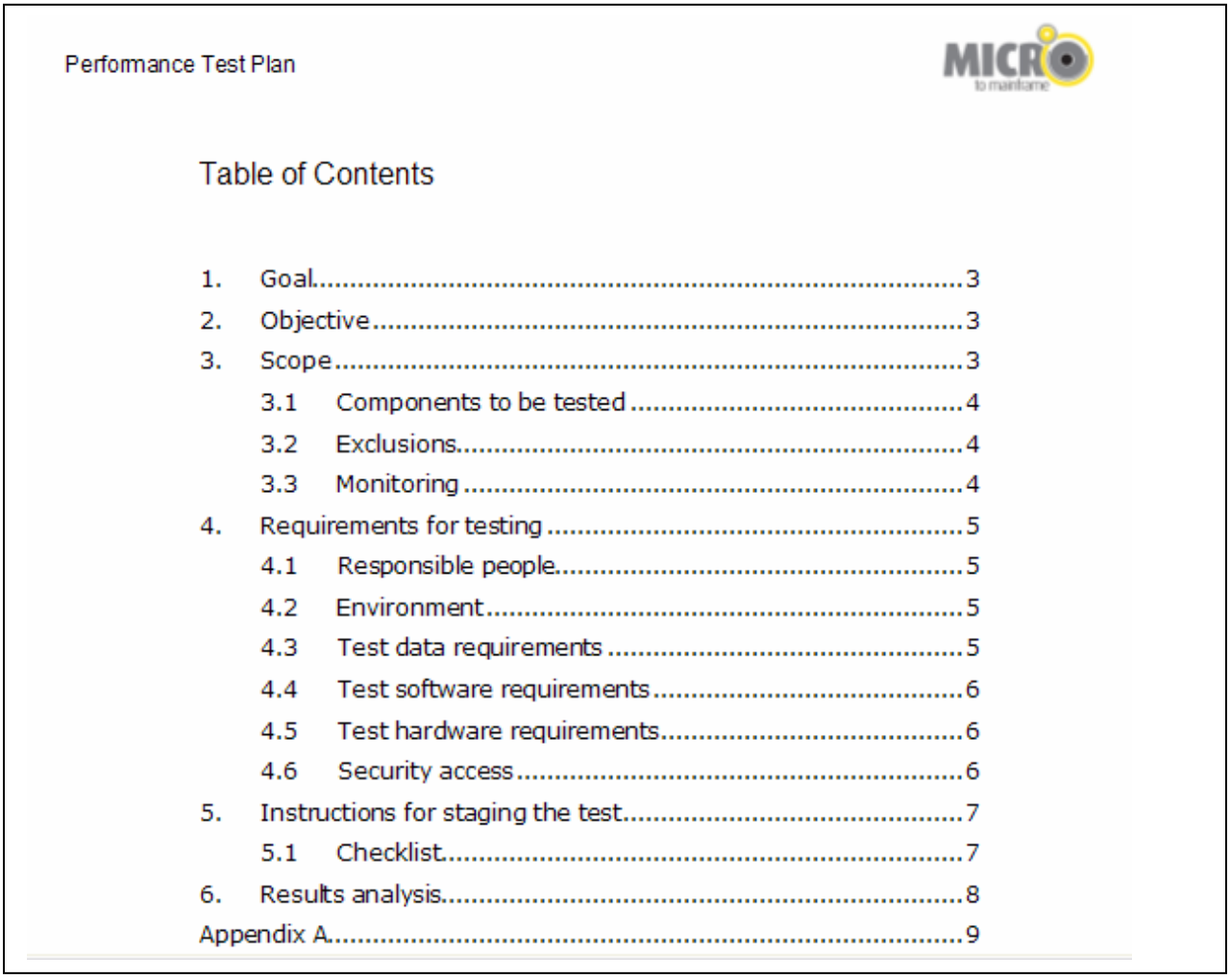

Figure 3

Phase 3 - Scripting

When you find problems during initial testing, the test tool and scripts are always blamed first. It is essential that you are 100% sure that your scripts do not cause any problems or errors on the system. Understanding the tool and how it interacts with the system is just as important. You need to be in a position where you can completely trust the tool and your scripts. You need to be able to confirm this and at the same time gain the respect of the necessary people.

"You need to be in a position where you can completely trust the tool and your scripts"

One of the biggest challenges for a performance tester is to win the respect of the people outside of testing. This includes developers, system architects and administrators such as database and network administrators. Any person that is responsible for the wellbeing of the system or application is affected by the performance test results. If you have these people on your side and win their respect early on, you can look forward to a good project.

Scripting is where the testing starts. It is extremely important that you familiarise yourself 100% with the application. A good tester will get a feel for the system right from the start. As you learn the system and processes to script, make notes of response times and slow or very busy processes. All of these may be potential bottlenecks in the system. When you start executing scripts, monitor the processes you identified closely and you may identify the first problem before any formal performance or load test was done.

"A good tester will get a feel for the system"

Monitoring starts during the scripting phase as well. When running a script for the first time the response times for measured transactions should be noted already. Keeping track of these response times is very important as changes in system behavior can be spotted by running one script only and save the time of preparing and setting up full load tests. Always run individual scripts at least once after every system change or implementation. *Figure 4* shows an example of a response time in the replay log of a script. In *Figure 5* the same response can be seen after a change that had a negative effect on the system. Identifying this eliminates the need for further tests after this change as performance clearly got worse or something with the implementation is wrong.

Keeping an eye on everything every time an individual script is run saves a lot of time and trouble. There is nothing worse than starting a full-blown load test with unique data for multiple users in place only to find that something is not right and the whole data and scenario setup have to be repeated. Attention to detail and understanding system behavior help to avoid these situations.

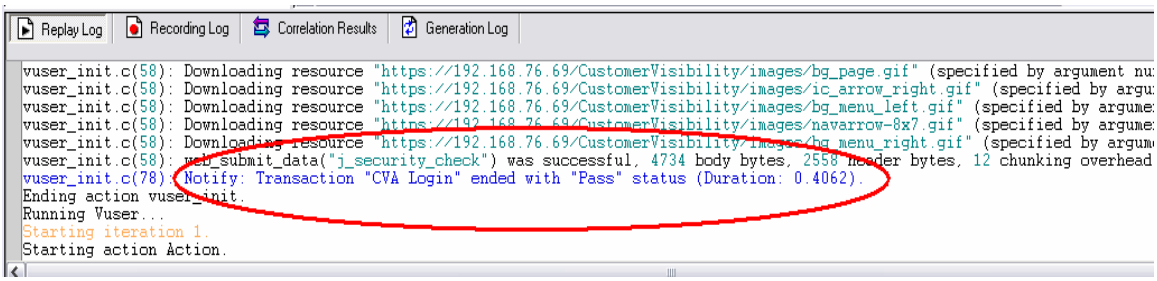

Figure 4 – Response time before change

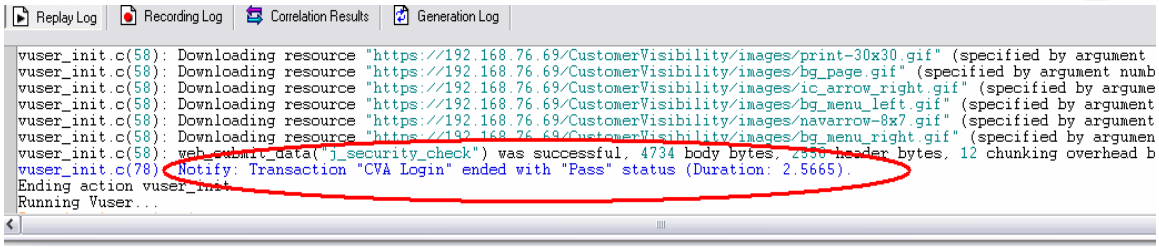

Figure 5 – Response time after change with negative impact

Best practices to remember during scripting include the following:

- Confirm application is scriptable
- Familiarise 100% with application
- Understand the business processes
- Understand the data requirements
- Know the environment
- Get a feel for the system's behavior
- Make sure that scripts exercise the whole environment
- Add response time measurements for every step
- Take note of response times from the first script replay onwards
- Verify script execution in the database
- Manage test data properly
- Run individual scripts at least once after every change or implementation

Phase 4 – Test Execution

There are different views and ways of talking about performance testing. Most people talk about load testing or stress testing. Although there isn't a definite right or wrong I refer to performance testing as a whole and this includes the testing as described in *Table 3*. The methodology was developed using this terminology.

Performance Testing	
Type of Test	**Description**
Baseline test	Establish performance baselines
Load test	Emulate production load on the system
Stress test	Load the system to breakpoint
Soak test	Test the system over a long period of time
Volume test	High data volumes / throughput. Database growth

Table 3

Baseline tests

"85% of performance problems can be identified and solved during baseline test runs"

Baseline tests are often mentioned but also ignored. However, they hold far more value than just establishing performance baselines and are one of the most important steps in this methodology. With some effort and time taken to examine details, up to 85% of performance problems can be identified and solved during baseline test runs. Unfortunately there is often not enough time for proper baseline testing, so it is important to include and plan for baseline tests right from the beginning of a project.

Baseline tests are done with each script individually. Typically each script is run with 1, 2, 5, 10 and 20 users. The maximum number of users will differ from project to project and is also dependent on the type of transaction or business process scripted. In some cases 5 or 10 users may be the maximum for a specific script.

Full monitoring should be done during the baseline testing. All results must be saved and analysed. The advantage of this is that all measurements are specific to the one process or transaction and problems can be identified without the trouble of isolating the process causing the problem or error. *Figure 6* shows the response times of one script. The example shown is the result of a 20-user test that caused very high CPU utilization, *Figure 7*. The high CPU utilization only happened with the one script.

"The advantage of this is that all measurements are specific to the one process"

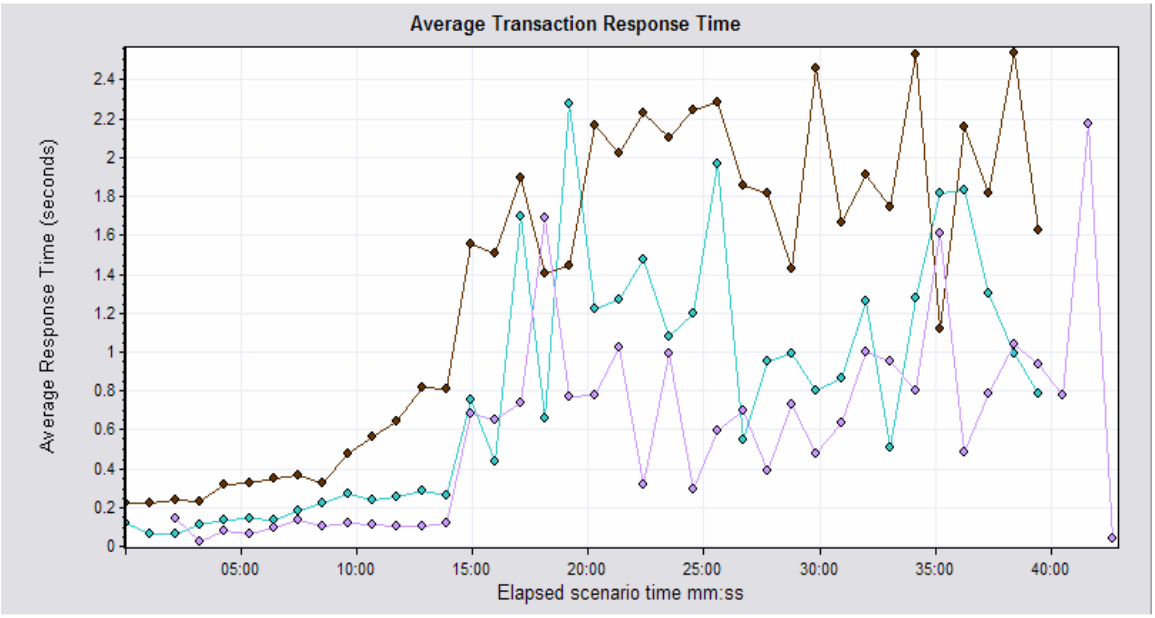

Figure 6 – Response time graph for one script with 3 measurements

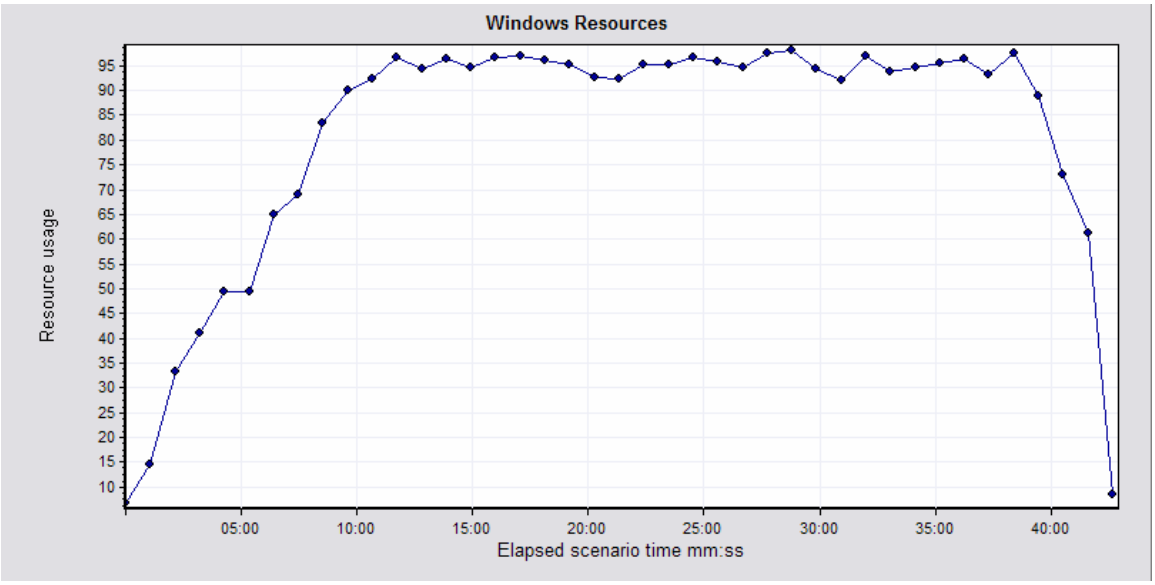

Figure 7 – CPU utilization very high for 20 users performing a single process

The problem was identified during baseline testing and isolated to the one script. No time was wasted for setting up and preparing for a full load test with multiple scripts and then spending

more time trying to isolate the cause of high CPU from a typical load test response time graph as can be seen in *Figure 8*. The objective is to iron out problems early during baseline testing and have an almost "clear run" when the first full load test is done. This saves everyone the frustration of trying to establish the cause of a problem or bottleneck with numerous users and transactions all being monitored at the same time.

"The objective is to have an almost 'clear run' when the first load test is done"

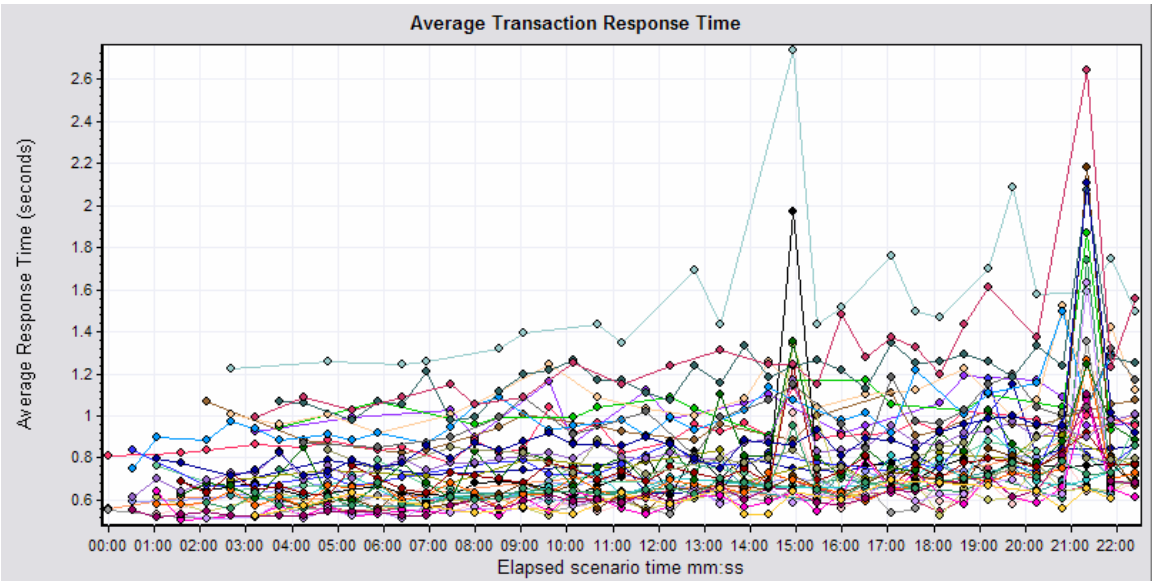

Figure 8 – Isolating problems between many processes can be impossible

Following the methodology with all baseline tests completed before the load test runs begin, ensures that every test run produces meaningful results. This is a very important aspect of the test process as early, meaningful results make people see your efforts as well as the value you are adding right from the start.

"Early, meaningful results make people see your efforts"

Load tests

When I started my performance testing career, I was trained how to plan, script and execute load tests. Although scripts were run individually during the scripting phase and for data preparation, the aim was to prepare for a load test. One of the main drawbacks of this was that the first test results were only available after a long period of preparation, often without meaningful or even readable results. Although problems were identified by this method, it was very difficult and time consuming to determine the cause of the problem. Add to this the very "busy" graphs from load test results and it becomes clear why a more productive and meaningful methodology should be followed.

These factors resulted in the development of a new methodology that delivered quick and meaningful results. Load testing is now only a part of the performance testing project and most problems are identified and solved before the first load test run.

"Most problems are identified and solved before the first load test run"

Stress test

A stress test is run to determine the breakpoint of the system. This should be done once all problems that stem from the performance testing have been resolved. The results of a stress test can be used for capacity planning and it gives administrators a view of how the system breaks and recovers. Plan to include at least one stress test towards the end of the project.

Soak test

A soak test is a load test that runs over a long period of time. Memory leaks are probably the most common problem to look for during a soak test, but connection timeouts are often found and database growth can also be monitored. Include all the relevant people when planning the soak test and ask them what they want monitored for their specific area.

Time and other logistical issues such as test data are some of the main problems to overcome when planning and executing a proper soak test. The test should run as long as possible, or until a specific trend can be identified through some of the monitors. If any serious defects are present, this will most probably determine the duration of the soak test.

Volume test

"Database size can influence performance greatly"

Volume testing refers to size and more specific database and file sizes. This is not always a requirement, but can be important depending on the type of application being tested. Database size can influence performance greatly and this should be put forward as a risk if the performance testing is done against a small database compared to what the size would be in production.

With volume testing load might not be required in the form of high user numbers and careful planning of how to execute the test is needed.

Phase 5 – Results Analysis

"The results that you want from a test must be the goal to work for when designing performance test scenarios"

Results analysis is perhaps the most challenging aspect of performance testing. It starts with the design of scenarios and tests that will give you the right "picture" when you look at the results at the end of a test run. Meaningful results are not always achieved and I believe the results that you want from a test must be the goal to work for when designing performance test scenarios.

Test results are the most important deliverable for the performance tester. This is after all the most effective way of showing the success of the testing. At the beginning of the paper I mentioned the first and the last test results from a project. This is the goal to work for. The comparison between the first and the last test run. How much improvement is shown as a result of the performance testing?

There are two main rules to follow to ensure successful results analysis:

1. Save everything.
   Name and save the results of every test run you do. Use a proper and sensible naming convention to make referencing at a later stage easy.

2. Keep track of everything.
   Make notes on why a test fails or why performance is poor or different than before. Also keep notes of why results are good or better than before. What was changed? Add the changes and the effect they have on system performance to the results summary after each run. Save it all for later reference and to use in the final test report.

Performance testing is an iterative process with many test runs. A short results summary is the most effective way to communicate results between test runs and most often the time between test runs is not enough to compile a full test report. The summary documents are good physical deliverables that make your effort more visible to the people making the investment. The results summary includes the following:

- Overview of test
- Scenario summary
- Number of users
- Maximum users
- Duration
- Total throughput (Bytes)
- Total hits
- Average hits per second
- Graphs
- Response time graphs
- System resource graphs
- Comparison graphs
- Recommendations for next test

Phase 6 – Reporting

The last methodology phase is to report back on the findings and progress of the whole project. A full performance test report is delivered with a presentation to communicate the content of the report to the relevant people.

"A report on its own is not very effective"

The aim is to explain the content of the final report and answer questions anyone might have about the testing and findings. I have learnt through experience that a report on its own is not very effective and most people never read it. Do both the report and presentation in a manner that non-technical people can understand it as well.

The final report doesn't refer to the results of one specific test and covers the findings of the test process as a whole. Graphs are included mainly for comparison with the emphasis on performance improvement throughout the project. Detailed results are not included but a reference to the relevant results summaries are given where specific issues are discussed.

Summary

The methodology described in this paper has been proven on various projects with various architectures. It is by no means the only methodology that works, but it does give you assurance of positive results. Visibility of the value added and the guarantee of success are the main reasons for developing and implementing this methodology.

Contact Details

The author:

Johann du Plessis
Telephone number: 27 76 601 4840
Email: johannd@mtom.co.za

Micro to Mainframe Offices:

Telephone numbers	
General:	27 11 547 7900
Talent acquisition:	27 11 547 7902
Human Resources:	27 11 547 7905
Admin office:	27 11 547 7921
Mentor to Mastery:	27 11 547 7908
Cape Town office:	27 21 789 1543
Fax number:	086 684 7176 / 011 484 9154

Physical address
1st Floor Block C
Metropolitan Park
8 Hillside Road
Parktown
South Africa

Postal address
PO Box 167749
Brackendowns
1454

Email addresses	
General:	enquiries@mtom.co.za
Careers at MtoM:	careers@mtom.co.za
Mentor to Mastery:	mentor@mtom.co.za
Client Relationship Manager:	clientcare@mtom.co.za
Accounts:	accounts@mtom.co.za