Chapter 4 – Web Application Performance Testing Core Activities

1

Performance Testing Guidance for Web Applications

J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, and Dennis Rea
Microsoft Corporation

September 2007

Objectives

- Learn the seven core activities that are integral to the majority of performance-testing projects.
- Understand the seven core activities in sufficient detail to identify how your tasks and processes map to these activities.
- Understand various performance-testing approaches that can be built around the core activities.

Overview

This chapter provides a high-level introduction to the most common core activities involved in performance-testing your applications and the systems that support those applications. Performance testing is a complex activity that cannot effectively be shaped into a "one-size-fits-all" or even a "one-size-fits-most" approach. Projects, environments, business drivers, acceptance criteria, technologies, timelines, legal implications, and available skills and tools simply make any notion of a common, universal approach unrealistic.

That said, there are some activities that are part of nearly all project-level performance-testing efforts. These activities may occur at different times, be called different things, have different degrees of focus, and be conducted either implicitly or explicitly, but when all is said and done, it is quite rare when a performance-testing-project does not involve at least making a decision around the seven core activities identified and referenced throughout this guide. These seven core activities do not in themselves constitute an approach to performance testing; rather, they represent the foundation upon which an approach can be built that is appropriate for your project.

How to Use This Chapter

Use this chapter to understand the core activities of performance testing and what these activities accomplish. To get the most from this chapter:

- Use the "Summary Table of Core Performance Activities" section to get an overview of the core activities of performance testing, and as quick reference guide for you and your team.
- Use the various Activity sections to understand the details of the most critical performance-testing tasks, as well as considerations for each activity.

Overview of Activities

The following sections discuss the seven activities that most commonly occur across successful performance-testing projects. The key to effectively implementing these activities is not when you conduct them, what you call them, whether or not they overlap, or the iteration pattern among them, but rather that you understand and carefully consider the concepts, applying them in the manner that is most valuable to your own project context.

Starting with at least a cursory knowledge of the project context, most teams begin identifying the test environment and the performance acceptance criteria more or less in parallel. This is due to the fact that all of the remaining activities are affected by the information gathered in activities 1 and 2. Generally, you will revisit these activities periodically as you and your team learn more about the application, its users, its features, and any performance-related risks it might have.

Once you have a good enough understanding of the project context, the test environment, and the performance acceptance criteria, you will begin planning and designing performance tests and configuring the test environment with the tools needed to conduct the kinds of performance tests and collect the kinds of data that you currently anticipate needing, as described in activities 3 and 4. Once again, in most cases you will revisit these activities periodically as more information becomes available.

With at least the relevant aspects of activities 1 through 4 accomplished, most teams will move into an iterative test cycle (activities 5-7) where designed tests are implemented by using some type of load-generation tool, the implemented tests are executed, and the results of those tests are analyzed and reported in terms of their relation to the components and features available to test at that time.

To the degree that performance testing begins before the system or application to be tested has been completed, there is a naturally iterative cycle that results from testing features and components as they become available and continually gaining more information about the application, its users, its features, and any performance-related risks that present themselves via testing.

Summary Table of Core Performance-Testing Activities

The following table summarizes the seven core performance-testing activities along with the most common input and output for each activity. Note that project context is not listed, although it is a critical input item for each activity.

Activity	**Input**	**Output**
Activity 1. Identify the Test Environment	• Logical and physical production architecture • Logical and physical test architecture • Available tools	• Comparison of test and production environments • Environment-related concerns • Determination of whether additional tools are required
Activity 2. Identify Performance Acceptance Criteria	• Client expectations • Risks to be mitigated • Business requirements • Contractual obligations	• Performance-testing success criteria • Performance goals and requirements • Key areas of investigation • Key performance indicators • Key business indicators
	• Available application	• Conceptual strategy

Activity 3. Plan and Design Tests	features and/or components • Application usage scenarios • Unit tests • Performance acceptance criteria	• Test execution prerequisites • Tools and resources required • Application usage models to be simulated • Test data required to implement tests • Tests ready to be implemented
Activity 4. Configure the Test Environment	• Conceptual strategy • Available tools • Designed tests	• Configured load-generation and resource-monitoring tools • Environment ready for performance testing
Activity 5. Implement the Test Design	• Conceptual strategy • Available tools/environment • Available application features and/or components • Designed tests	• Validated, executable tests • Validated resource monitoring • Validated data collection
Activity 6. Execute the Test	• Task execution plan • Available tools/environment • Available application features and/or components • Validated, executable tests	• Test execution results
Activity 7. Analyze Results, Report, and Retest	• Task execution results • Performance acceptance criteria • Risks, concerns, and issues	• Results analysis • Recommendations • Reports

Core Performance-Testing Activities Walkthrough

The seven core performance-testing activities can be summarized as follows.

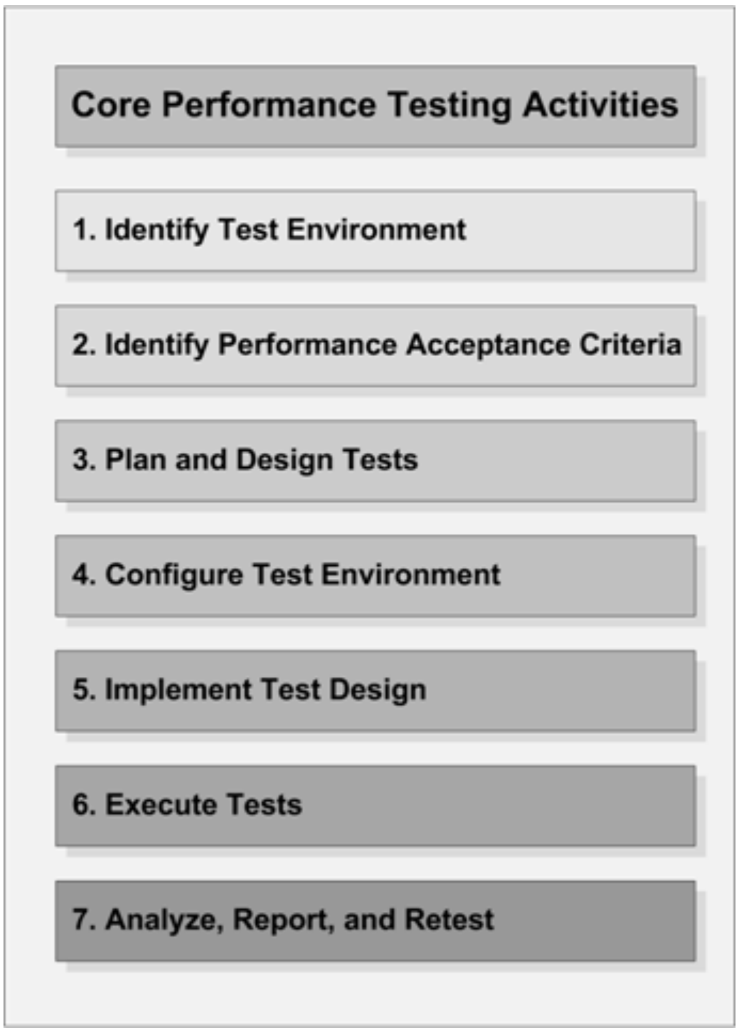
Core Performance Testing Activities
1. Identify Test Environment
2. Identify Performance Acceptance Criteria
3. Plan and Design Tests
4. Configure Test Environment
5. Implement Test Design
6. Execute Tests
7. Analyze, Report, and Retest

Figure 4.1 *Core Performance Testing Activities*

1. **Activity 1. Identify the Test Environment.** Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software, and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. In some situations, this process must be revisited periodically throughout the project's life cycle.
2. **Activity 2. Identify Performance Acceptance Criteria.** Identify the response time, throughput, and resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern, and resource utilization is a system concern. Additionally, identify project success criteria that may not be captured by those goals and constraints; for example, using performance tests to evaluate what combination of configuration settings will result in the most desirable performance characteristics.
3. **Activity 3. Plan and Design Tests.** Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected. Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed.
4. **Activity 4. Configure the Test Environment.** Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test. Ensure that the test environment is instrumented for resource monitoring as necessary.
5. **Activity 5. Implement the Test Design.** Develop the performance tests in accordance with the test design.
6. **Activity 6. Execute the Test.** Run and monitor your tests. Validate the tests, test data, and results collection. Execute validated tests for analysis while monitoring the test and the test environment.

7. **Activity 7. Analyze Results, Report, and Retest.** Consolidate and share results data. Analyze the data both individually and as a cross-functional team. Reprioritize the remaining tests and re-execute them as needed. When all of the metric values are within accepted limits, none of the set thresholds have been violated, and all of the desired information has been collected, you have finished testing that particular scenario on that particular configuration.

Activity 1. Identify the Test Environment

The environment in which your performance tests will be executed, along with the tools and associated hardware necessary to execute the performance tests, constitute the test environment. Under ideal conditions, if the goal is to determine the performance characteristics of the application in production, the test environment is an exact replica of the production environment but with the addition of load-generation and resource-monitoring tools. Exact replicas of production environments are uncommon.

The degree of similarity between the hardware, software, and network configuration of the application under test conditions and under actual production conditions is often a significant consideration when deciding what performance tests to conduct and what size loads to test. It is important to remember that it is not only the physical and software environments that impact performance testing, but also the objectives of the test itself. Often, performance tests are applied against a proposed new hardware infrastructure to validate the supposition that the new hardware will address existing performance concerns.

The key factor in identifying your test environment is to completely understand the similarities and differences between the test and production environments. Some critical factors to consider are:

- Hardware
  - Configurations
  - Machine hardware (processor, RAM, etc.)
- Network
  - Network architecture and end-user location
  - Load-balancing implications
  - Cluster and Domain Name System (DNS) configurations
- Tools
  - Load-generation tool limitations
  - Environmental impact of monitoring tools
- Software
  - Other software installed or running in shared or virtual environments
  - Software license constraints or differences
  - Storage capacity and seed data volume
  - Logging levels
- External factors
  - Volume and type of additional traffic on the network
  - Scheduled or batch processes, updates, or backups
  - Interactions with other systems

Considerations

Consider the following key points when characterizing the test environment:

- Although few performance testers install, configure, and administrate the application being tested, it is beneficial for the testers to have access to the servers and software, or to the administrators who do.
- Identify the amount and type of data the application must be seeded with to emulate real-world conditions.
- Identify critical system components. Do any of the system components have known performance concerns? Are there any integration points that are beyond your control for testing?
- Get to know the IT staff. You will likely need their support to perform tasks such as monitoring

overall network traffic and configuring your load-generation tool to simulate a realistic number of Internet Protocol (IP) addresses.
- Check the configuration of load balancers.
- Validate name resolution with DNS. This may account for significant latency when opening database connections.
- Validate that firewalls, DNS, routing, and so on treat the generated load similarly to a load that would typically be encountered in a production environment.
- It is often appropriate to have systems administrators set up resource-monitoring software, diagnostic tools, and other utilities in the test environment.

Activity 2. Identify Performance Acceptance Criteria

It generally makes sense to start identifying, or at least estimating, the desired performance characteristics of the application early in the development life cycle. This can be accomplished most simply by noting the performance characteristics that your users and stakeholders equate with good performance. The notes can be quantified at a later time.

Classes of characteristics that frequently correlate to a user's or stakeholder's satisfaction typically include:

- **Response time.** For example, the product catalog must be displayed in less than three seconds.
- **Throughput.** For example, the system must support 25 book orders per second.
- **Resource utilization.** For example, processor utilization is not more than 75 percent. Other important resources that need to be considered for setting objectives are memory, disk input/output (I/O), and network I/O.

Considerations

Consider the following key points when identifying performance criteria:

- Business requirements
- User expectations
- Contractual obligations
- Regulatory compliance criteria and industry standards
- Service Level Agreements (SLAs)
- Resource utilization targets
- Various and diverse, realistic workload models
- The entire range of anticipated load conditions
- Conditions of system stress
- Entire scenarios and component activities
- Key performance indicators
- Previous releases of the application
- Competitor's applications
- Optimization objectives
- Safety factors, room for growth, and scalability
- Schedule, staffing, budget, resources, and other priorities

Activity 3. Plan and Design Tests

Planning and designing performance tests involves identifying key usage scenarios, determining appropriate variability across users, identifying and generating test data, and specifying the metrics to be collected. Ultimately, these items will provide the foundation for workloads and workload profiles.

When designing and planning tests with the intention of characterizing production performance, your goal should be to create real-world simulations in order to provide reliable data that will enable your organization to make informed business decisions. Real-world test designs will significantly increase the relevancy and usefulness of results data.

Key usage scenarios for the application typically surface during the process of identifying the desired performance characteristics of the application. If this is not the case for your test project, you will need to explicitly determine the usage scenarios that are the most valuable to script. Consider the following when identifying key usage scenarios:

- Contractually obligated usage scenario(s)
- Usage scenarios implied or mandated by performance-testing goals and objectives
- Most common usage scenario(s)
- Business-critical usage scenario(s)
- Performance-intensive usage scenario(s)
- Usage scenarios of technical concern
- Usage scenarios of stakeholder concern
- High-visibility usage scenarios

When identified, captured, and reported correctly, metrics provide information about how your application's performance compares to your desired performance characteristics. In addition, metrics can help you identify problem areas and bottlenecks within your application.

It is useful to identify the metrics related to the performance acceptance criteria during test design so that the method of collecting those metrics can be integrated into the tests when implementing the test design. When identifying metrics, use either specific desired characteristics or indicators that are directly or indirectly related to those characteristics.

Considerations

Consider the following key points when planning and designing tests:

- Realistic test designs are sensitive to dependencies outside the control of the system, such as humans, network activity, and other systems interacting with the application.
- Realistic test designs are based on what you expect to find in real-world use, not theories or projections.
- Realistic test designs produce more credible results and thus enhance the value of performance testing.
- Component-level performance tests are integral parts of realistic testing.
- Realistic test designs can be more costly and time-consuming to implement, but they provide far more accuracy for the business and stakeholders.
- Extrapolating performance results from unrealistic tests can create damaging inaccuracies as the system scope increases, and frequently lead to poor decisions.
- Involve the developers and administrators in the process of determining which metrics are likely to add value and which method best integrates the capturing of those metrics into the test.
- Beware of allowing your tools to influence your test design. Better tests almost always result from designing tests on the assumption that they can be executed and then adapting the test or the tool when that assumption is proven false, rather than by *not* designing particular tests based on the assumption that you do not have access to a tool to execute the test.

Realistic test designs include:

- Realistic simulations of user delays and think times, which are crucial to the accuracy of the test.
- User abandonment, if users are likely to abandon a task for any reason.
- Common user errors.

Activity 4. Configure the Test Environment

Preparing the test environment, tools, and resources for test design implementation and test execution prior to features and components becoming available for test can significantly increase the amount of testing that can be accomplished during the time those features and components are available.

Load-generation and application-monitoring tools are almost never as easy to get up and running as one expects. Whether issues arise from setting up isolated network environments, procuring hardware, coordinating a dedicated bank of IP addresses for IP spoofing, or version compatibility between monitoring software and server operating systems, issues always seem to arise from somewhere. Start early, to ensure that issues are resolved before you begin testing.

Additionally, plan to periodically reconfigure, update, add to, or otherwise enhance your load-generation environment and associated tools throughout the project. Even if the application under test stays the same and the load-generation tool is working properly, it is likely that the metrics you want to collect will change. This frequently implies some degree of change to, or addition of, monitoring tools.

Considerations

Consider the following key points when configuring the test environment:

- Determine how much load you can generate before the load generators reach a bottleneck. Typically, load generators encounter bottlenecks first in memory and then in the processor.
- Although it may seem like a commonsense practice, it is important to verify that system clocks are synchronized on all of the machines from which resource data will be collected. Doing so can save you significant time and prevent you from having to dispose of the data entirely and repeat the tests after synchronizing the system clocks.
- Validate the accuracy of load test execution against hardware components such as switches and network cards. For example, ensure the correct full-duplex mode operation and correct emulation of user latency and bandwidth.
- Validate the accuracy of load test execution related to server clusters in load-balanced configuration. Consider using load-testing techniques to avoid affinity of clients to servers due to their using the same IP address. Most load-generation tools offer the ability to simulate usage of different IP addresses across load-test generators.
- Monitor resource utilization (CPU, network, memory, disk and transactions per time) across servers in the load-balanced configuration during a load test to validate that the load is distributed.

Activity 5. Implement the Test Design

The details of creating an executable performance test are extremely tool-specific. Regardless of the tool that you are using, creating a performance test typically involves scripting a single usage scenario and then enhancing that scenario and combining it with other scenarios to ultimately represent a complete workload model.

Load-generation tools inevitably lag behind evolving technologies and practices. Tool creators can only build in support for the most prominent technologies and, even then, these have to become prominent before the support can be built. This often means that the biggest challenge involved in a performance-testing project is getting your first relatively realistic test implemented with users generally being simulated in such a way that the application under test cannot legitimately tell the difference between the simulated users and real users. Plan for this and do not be surprised when it takes significantly longer than expected to get it all working smoothly.

Considerations

Consider the following key points when implementing the test design:

- Ensure that test data feeds are implemented correctly. Test data feeds are data repositories in the form of databases, text files, in-memory variables, or spreadsheets that are used to simulate parameter replacement during a load test. For example, even if the application database test repository contains the full production set, your load test might only need to simulate a subset of products being bought by users due to a scenario involving, for example, a new product or marketing campaign. Test data feeds may be a subset of production data repositories.
- Ensure that application data feeds are implemented correctly in the database and other application components. Application data feeds are data repositories, such as product or order databases, that are consumed by the application being tested. The key user scenarios, run by the load test scripts

may consume a subset of this data.

- Ensure that validation of transactions is implemented correctly. Many transactions are reported successful by the Web server, but they fail to complete correctly. Examples of validation are, database entries inserted with correct number of rows, product information being returned, correct content returned in html data to the clients etc.
- Ensure hidden fields or other special data are handled correctly. This refers to data returned by Web server that needs to be resubmitted in subsequent request, like session IDs or product ID that needs to be incremented before passing it to the next request.
- Validate the monitoring of key performance indicators (KPIs).
- Add pertinent indicators to facilitate articulating business performance.
- If the request accepts parameters, ensure that the parameter data is populated properly with variables and/or unique data to avoid any server-side caching.
- If the tool does not do so automatically, consider adding a wrapper around the requests in the test script in order to measure the request response time.
- It is generally worth taking the time to make the script match your designed test, rather than changing the designed test to save scripting time.
- Significant value can be gained from evaluating the output data collected from executed tests against expectations in order to test or validate script development.

Activity 6. Execute the Test

Executing tests is what most people envision when they think about performance testing. It makes sense that the process, flow, and technical details of test execution are extremely dependent on your tools, environment, and project context. Even so, there are some fairly universal tasks and considerations that need to be kept in mind when executing tests.

Much of the performance testing–related training available today treats test execution as little more than starting a test and monitoring it to ensure that the test appears to be running as expected. In reality, this activity is significantly more complex than just clicking a button and monitoring machines.

Test execution can be viewed as a combination of the following sub-tasks:

1. Coordinate test execution and monitoring with the team.
2. Validate tests, configurations, and the state of the environments and data.
3. Begin test execution.
4. While the test is running, monitor and validate scripts, systems, and data.
5. Upon test completion, quickly review the results for obvious indications that the test was flawed.
6. Archive the tests, test data, results, and other information necessary to repeat the test later if needed.
7. Log start and end times, the name of the result data, and so on. This will allow you to identify your data sequentially after your test is done.

As you prepare to begin test execution, it is worth taking the time to double-check the following items:

- Validate that the test environment matches the configuration that you were expecting and/or designed your test for.
- Ensure that both the test and the test environment are correctly configured for metrics collection.
- Before running the real test, execute a quick smoke test to make sure that the test script and remote performance counters are working correctly. In the context of performance testing, a *smoke test* is designed to determine if your application can successfully perform all of its operations under a normal load condition for a short time.
- Reset the system (unless your scenario calls for doing otherwise) and start a formal test execution.
- Make sure that the test scripts’ execution represents the workload model you want to simulate.
- Make sure that the test is configured to collect the key performance and business indicators of interest at this time.

Considerations

Consider the following key points when executing the test:

- Validate test executions for data updates, such as orders in the database that have been completed.
- Validate if the load-test script is using the correct data values, such as product and order identifiers, in order to realistically simulate the business scenario.
- Whenever possible, limit test execution cycles to one to two days each. Review and reprioritize after each cycle.
- If at all possible, execute every test three times. Note that the results of first-time tests can be affected by loading Dynamic-Link Libraries (DLLs), populating server-side caches, or initializing scripts and other resources required by the code under test. If the results of the second and third iterations are not highly similar, execute the test again. Try to determine what factors account for the difference.
- Observe your test during execution and pay close attention to any behavior you feel is unusual. Your instincts are usually right, or at least valuable indicators.
- No matter how far in advance a test is scheduled, give the team 30-minute and 5-minute warnings before launching the test (or starting the day's testing) if you are using a shared test environment. Additionally, inform the team whenever you are not going to be executing for more than one hour in succession so that you do not impede the completion of their tasks.
- Do not process data, write reports, or draw diagrams on your load-generating machine while generating a load, because this can skew the results of your test.
- Turn off any active virus-scanning on load-generating machines during testing to minimize the likelihood of unintentionally skewing the results of your test.
- While load is being generated, access the system manually from a machine outside of the load-generation environment during test execution so that you can compare your observations with the results data at a later time.
- Remember to simulate ramp-up and cool-down periods appropriately.
- Do not throw away the first iteration because of application script compilation, Web server cache building, or other similar reasons. Instead, measure this iteration separately so that you will know what the first user after a system-wide reboot can expect.
- Test execution is never really finished, but eventually you will reach a point of diminishing returns on a particular test. When you stop obtaining valuable information, move on to other tests.
- If you feel you are not making progress in understanding an observed issue, it may be more efficient to eliminate one or more variables or potential causes and then run the test again.

Activity 7. Analyze Results, Report, and Retest

Managers and stakeholders need more than just the results from various tests — they need conclusions, as well as consolidated data that supports those conclusions. Technical team members also need more than just results — they need analysis, comparisons, and details behind how the results were obtained. Team members of all types get value from performance results being shared more frequently.

Before results can be reported, the data must be analyzed. Consider the following important points when analyzing the data returned by your performance test:

- Analyze the data both individually and as part of a collaborative, cross-functional technical team.
- Analyze the captured data and compare the results against the metric's acceptable or expected level to determine whether the performance of the application being tested shows a trend toward or away from the performance objectives.
- If the test fails, a diagnosis and tuning activity are generally warranted.
- If you fix any bottlenecks, repeat the test to validate the fix.
- Performance-testing results will often enable the team to analyze components at a deep level and correlate the information back to the real world with proper test design and usage analysis.
- Performance test results should enable informed architecture and business decisions.
- Frequently, the analysis will reveal that, in order to completely understand the results of a particular test, additional metrics will need to be captured during subsequent test-execution cycles.
- Immediately share test results and make raw data available to your entire team.

- Talk to the consumers of the data to validate that the test achieved the desired results and that the data means what you think it means.
- Modify the test to get new, better, or different information if the results do not represent what the test was defined to determine.
- Use current results to set priorities for the next test.
- Collecting metrics frequently produces very large volumes of data. Although it is tempting to reduce the amount of data, always exercise caution when using data-reduction techniques because valuable data can be lost.

Most reports fall into one of the following two categories:

- **Technical Reports**
  - Description of the test, including workload model and test environment.
  - Easily digestible data with minimal pre-processing.
  - Access to the complete data set and test conditions.
  - Short statements of observations, concerns, questions, and requests for collaboration.
- **Stakeholder Reports**
  - Criteria to which the results relate.
  - Intuitive, visual representations of the most relevant data.
  - Brief verbal summaries of the chart or graph in terms of criteria.
  - Intuitive, visual representations of the workload model and test environment.
  - Access to associated technical reports, complete data sets, and test conditions.
  - Summaries of observations, concerns, and recommendations.

The key to effective reporting is to present information of interest to the intended audience in a manner that is quick, simple, and intuitive. The following are some underlying principles for achieving effective reports:

- Report early, report often.
- Report visually.
- Report intuitively.
- Use the right statistics.
- Consolidate data correctly.
- Summarize data effectively.
- Customize for the intended audience.
- Use concise verbal summaries using strong but factual language.
- Make the data available to stakeholders.
- Filter out any unnecessary data.
- If reporting intermediate results, include the priorities, concerns, and blocks for the next several test-execution cycles.

Summary

Performance testing involves a set of common core activities that occur at different stages of projects. Each activity has specific characteristics and tasks to be accomplished. These activities have been found to be present — or at least to have been part of an active, risk-based decision to omit one of the activities — in every deliberate and successful performance-testing project that the authors and reviewers have experienced. It is important to understand each activity in detail and then apply the activities in a way that best fits the project context.

1

Links Table

[1] *http://www.microsoft.com/practices*

Community Content