

Load Testing Is Easy. Good Load Testing Is Not. Preparation is the difference.

Prepared by Chris Lynn
and Dennis Sherman

This paper discusses recommendations and considerations to plan and prepare for valuable performance and load testing. It includes a guide to identifying and developing the:

- 1. Testing Purposes*
- 2. Roles and Responsibilities*
- 3. Business Performance Requirements*
- 4. Scope of Testing*
- 5. Testing Environment Requirements*
- 6. Usage Patterns and Transaction Mix*
- 7. Proposed Test Scenarios*
- 8. Load Generation Requirements*
- 9. Proposed Monitoring, Tracking, and Reporting*

Once this planning is done the scripting and execution is “easy”. Scripting not included.

Introduction

Many people believe that load and performance testing is just as simple as running virtual user scripts against an application. That is true if it isn't important to interpret the results of the testing. Careful planning is necessary to ensure that the correct transactions are tested in an environment that will allow conclusions to be drawn about how the application being tested will perform in production. This paper provides a simple, repeatable approach to planning effective and efficient load testing. Included at the end of this paper are some worksheets and meeting templates associated with the concepts presented here.

Disclaimer: This is not a complete and all inclusive list of all possible scenarios, and concepts. It is an introductory planning guide to facilitate the right discussions and decisions in an efficient and effective manner. Sometimes steps can be skipped, and sometimes extra steps will be needed, but these guidelines will help keep you focused with a repeatable common starting point.

1. Testing Purposes

The required objectives or purposes of a load test are the most important things to determine. The purpose of the testing will dictate the scope of the testing, the testing environment, the load requirements and the test scenarios. Most testing will have several purposes, and some of the work to support each will overlap, but keep in mind that each additional purpose will require more resources, especially time. With that in mind you will want to ask: What are the “required purposes” of the test, not just the “nice to haves?”

There are several different common names for the purposes of load and performance testing. The names we use are listed and described below.

Performance Evaluation

The purpose of this test is to determine at what load (transaction types and volumes) the response times for requests start to degrade and then become unacceptable. This is almost always one of the purposes of any load test. This test

will also validate the test environment hardware, and if necessary provide you with the information necessary to model the results into production.

Performance Tuning

The purpose of this test is to identify performance bottlenecks. Once bottlenecks are identified tuning of the application or environment can be done to improve performance. This can be done to either improve application response times or to reduce resource consumption. Initial testing results may indicate that this purpose is required and additional tests should be conducted.

Stress

The purpose of this test is to determine the breaking point (maximum loads) of the application or current recommended hardware. Some times the objective will be to identify the breaking point of the application system as a whole, in other words the “first” break point. Other times testing should be done to identify the breaking point of each individual component of the architecture. The number of break points evaluated will dramatically affect the amount of time and effort required for the testing.

Availability

The purpose of this test is to determine if one redundant component of the architecture can handle the entire load in the event of the failure of its paired component, and to identify how the system will perform in the event of failures under load.

Stability

The purpose of this test is to determine if the hardware can support sustained loads. In other words, to verify that the system can not only support the peak load for seconds or minutes, but for hours or days at expected loads. In this scenario, you do not hit the server with peak load for days, but instead, you generate a large percentage of the peak load. As a rule of thumb, we typically use 65-75% of the anticipated peak loads for this type of test.

2. Roles and Responsibilities

When someone new to load testing begins a project that requires load testing they often believe it will be relatively simple and only involve the application testing team. However, in reality, a good load test always requires the involvement of several people filling vital roles. The purposes and the scope of testing will determine which roles are required, and what their responsibilities should be. Several common roles and corresponding responsibilities will be discussed here, but don't hesitate to include additional people if, depending on the specifics of your testing, it is merited. There are almost always adjustments to be made when load testing so the people in these roles should be skilled, creative, and adaptable. Each organization has its own names for performance testing roles so the following titles represent generic performance testing functions which should exist in most organizations.

Test Manager

The test manager coordinates the efforts of the overall process and the people involved in the load testing. Their first priority is to ensure that the purpose, scope, and success criteria of the testing are clearly defined. Next they ensure the test infrastructure and application setup is completed and at least meets the minimum environment requirements. Finally, they represent the customer throughout the load testing process to ensure that when the testing is complete all of the purposes have been addressed and the success criteria have been met. In many cases this role is also responsible for the other aspects of application testing (functional, integration, etc.), but those aspects are not included in this paper.

Load Tester

The load tester role is critical to the success of performance testing. In addition to writing scripts and running test scenarios a good load tester will also review the load testing plan, and help determine or validate the application usage patterns and peak loads. In order to define and execute a good load test this person has to have a sound understanding of the load testing tools that will be used including, how the options and configuration parameters (for example; # Virtual Users, think time, ramp up, etc.) can be used to best mimic the expected production load. The load

tester will also track the transaction volume and response time results. The load tester should also write load testing reports that are meaningful to the customer and others who are involved with the analysis of the tests.

They can also be a great contributor in any bottleneck or breaking point identification activities but should always rely on the experts in given areas to make any changes to the application. For example, a load tester may identify a transaction with unacceptable response times but they will want to consult with application developers and system administrators who can more effectively identify and address the cause of the problem.

Application/Business Representative

Someone who understands how the application works should be assigned to participate in the load testing process. This application knowledge is critical to determining what the transaction mix should be, as well as what the peak loads may be. The more accurate that data is the better the results of testing will be. The application representative does not need to understand the intricacies of load testing.

Only the application representative can provide what the application response time requirements should be. They also have the final acceptance of any testing results. Furthermore, only the application or business representative can anticipate what the future requirements will be for the application being tested. It doesn't make much sense to test for current loads if loads are expected to grow in the near future, or if the way the application is used today will change in the foreseeable future. That growth rate and any foreseeable changes in the business activity should be provided by the person in this role.

Capacity Planner

The capacity planner's role is to ensure that the load tests that are run will provide sufficient data to meet any planning objectives. They will help the load tester develop and design the testing parameters to ensure the tests will accurately simulate the production activity described by the application and business representatives. The capacity planning team will be especially interested in application response times, transaction rates, and how the various hardware and software systems respond to the various levels of load. They will be responsible for modeling the test results from the testing environment to the production environment to account for any discrepancies in processing power or load. Additionally, they may be required to model the testing results into the future to account for any growth that can't be tested for directly.

The capacity planning representative will likely provide a separate report, in addition to the load testing report, indicating whether or not the application met the stated business requirements and if it will be able to meet future needs based on business usage forecasts. They may also create additional models to account for any unexpected future circumstances, or to allow for inaccuracies due to assumptions made in the usage pattern or transaction rate planning.

Server Engineer

The server engineer is responsible for ensuring the environment being tested is a reasonable representation of the production environment where the application being tested will run. They ensure that the system is built with the same architecture as in production. Even if the size of the systems is not identical the architecture (same versions and interaction between components) needs to be. They will often build the environment. The person in this role is also responsible for ensuring the load generator(s) and load managers are installed correctly, and that all systems clocks are synchronized for comparison of results across those systems.

Network Engineer

Network impact should be considered as part of the load testing process, but frequently the testing reveals that the network is not a constrained resource. In other words you will always want to have a Network Engineer look at the network at some point in the testing, but seldom will they need to be involved in all of the testing. There are of course exceptions to this such as application testing over the WAN and any required application tuning. You should have a qualified network resource on the load testing team even if the network is not expected to be a throttle. It is less important to have exactly the same network resources in test as in production, but it will require a network engineer to really understand and model the differences. The Network Engineer can also be a great resource in anticipating the impact of tests over a LAN for an application that will be used over a WAN.

3. Business Performance Requirements

The business or the customer must define the requirements for the application being tested. It is critical that the load tester understand the business performance requirements (transaction volumes, response times and throughput). **Without this information, you can not know if the testing is successful.** If there are no defined requirements then *every test is successful*, and will never necessitate additional testing. These requirements will allow for a very technical discussion with predictable results rather than a potentially emotional response that the application doesn't perform "as hoped." Even "obvious" requirements should be collected. As with all requirements it's not to say that the business can't change the requirements as testing progresses, but getting an initial requirement will greatly reduce the amount of re-testing.

For example, if performance tuning is the purpose of the testing then what needs to be optimized, the transaction rate, the response time, or both? The business requirements for the application will dictate what needs to be worked on, and when the requirements are met, you are done testing. Or, if performance evaluation and availability are two of the purposes for testing, then it is probably a business requirement that the system perform equally well when a redundant portion of the environment fails.

4. Scope of Testing

What components will be evaluated as part of the testing? Some tests will require a detailed look at many of the components involved, while others will require a focus on only a few specific components. The time available for testing, the flexibility of the test environment, and the potential changes that can be made depending on the testing results should all be considered when determining what the scope should be.

Time

If you have a short time frame available then only the most critical components can be included. Even if time is not a limiting factor you will want to balance the value gained from testing with the time and effort required.

Potential Change Likelihood

If the current course of the application (deployment to production, performance capabilities, middleware choices, etc.) will not be altered as a result of testing a given set of components (perhaps based on standards, or policy, or politics), then there is little point in testing those components explicitly. For example, if WebSphere is your supported company standard and Weblogic would not be considered even if tests showed better performance then don't test the application on Weblogic.

Additional Considerations

Verify that the resources you use to generate the test load are not a limiting factor in your tests. If your load generation environment has constrained resources (for example: CPUs running at 100%), increase your load generation capabilities. Otherwise, test results indicating poor performance may be a reflection of your load generation environment, rather than of the application environment you are testing.

Also, do not forget to consider including the often forgotten batch processing, reporting, ETL, middleware, annual processing, alternate clients (wireless, time clocks) etc. when determining the proper scope for your tests.

Scope Examples

- 1) If you are regression testing some "look and feel" html changes in the application, then it is unlikely that application or database servers will be significantly impacted. So, they would not need to be included in the scope of this testing. Instead you should focus on response times of the client and perhaps resource utilization on the web server. Whether or not the network should be included would depend on if the web object sizes had changed significantly, and if the application were going to run over a constrained WAN, or an unconstrained LAN.
- 2) If you are testing a brand new application in your enterprise it would be valuable to identify the impact to all the systems supporting the new application. However, most applications do not stress all aspects of all resources and

the focus can be adjusted to only those most impacted areas as testing proceeds. For example, some applications are very disk intensive and the storage systems should be monitored closely, but in other cases the CPU is heavily stressed and the storage systems are not. It does not make sense to spend the time or resources to focus on everything all the time just because you can.

- 3) If your application has a low user volume, but significant backend (application/database) processing, then less focus can be placed on the web servers.
- 4) If you are simply tuning an application for response time improvements and the physical resources are not a throttle, then they do not need to be included in the scope of that testing.

5. Testing Environment Requirements

Testing environment requirements will vary depending on budget, timelines, what is being tested, testing purposes, etc. This testing is often executed in QA or development environments. As with most things, typically, for a given test, there is an ideal environment, a minimum required environment and other options somewhere in between. Ideally the test environment would be identical to production but rarely is that practical or cost effective. The minimum would be the smallest environment that will contain all architecturally significant components of the application and enable the test purposes to be met. For example if you have a three tiered system in production (web/app/database), the minimum may be as simple as a single box that has all three components. Some considerations for the required test environment are:

Hardware/Platform

The hardware and platform should be the same. You do not want to load test against a Windows environment if your production environment is AIX. You also want to ensure that the OS and patch versions are the same.

Load Balancing

You will need to know if you are testing through load balancers. If you test through a load balancer, you'll need to know the configuration details; for example, whether or not session affinity is enabled, and the load balancing method used such as round-robin, hash, or least connections. If you do not understand how the load is being distributed it is very easy to misinterpret the test results.

Network Simulation

Understanding the physical location of the application users relative to the application, from a networking perspective, will be critical to designing the tests and interpreting them correctly. If users are on a LAN then tests should be done on a LAN. If the users will connect to the application over the internet or over a WAN then ideally the tests should also be done over similar network connections. However, that is often not practical and consideration should be made as you interpreting response time results when a WAN application is tested in a LAN environment. If you do decide to simulate the network be sure to account for latency and errors in addition to bandwidth.

Software Infrastructure

An easily overlooked consideration is making sure that software versions match. You'll want to work with system administrators to ensure that operating systems, HTTP servers, database versions, application servers, java versions, ODBC drivers and all other supporting software use the production versions. Also, be sure the patch levels are the same in test as they are in production.

Application Functionality

Ideally the application will be 100% functional before you begin load testing. This isn't always practical when working to meet testing timelines, but the primary application functionality should have already passed functional testing before beginning load testing. When the application functions in the test environment the same way it will function in the production environment, the more accurate, and therefore more beneficial, the testing will be. Project managers will likely push to perform functional and load testing in parallel, but in our experience it just leads to wasted performance testing and a great deal of re-testing. Hint: If a single tester doing a functional test doesn't get acceptable performance

then performance can only get worse with more load. In other words the functional tester conducts the first very small load test.

Monitoring Tools

Monitoring tools are needed to document the load testing impact on the test servers as well as the load generation servers. Monitoring tools might include:

- Application troubleshooting tools like Wily Introscope
- Local system tools like Top/Topas, Perfmon
- Network tools like ACE, ethereal, snoop
- System metric agents like BMC Perform/Predict, CA Unicenter

This monitoring data will be crucial to modeling the test results into production, especially if the load tests are being run on an environment not identical to production.

6. Usage Patterns and Transaction Mix (Template # 1 – Transactions)

When deciding what transactions to load test it should be recognized that you do not necessarily need to every transaction. Load testing is not the same as functional testing where every possible use case should be examined. In a load test only the critical transactions should be tested. Through interviews with the business users, developers, and other data which may be available (for example, access logs) you will need to identify the most important transactions. Typically, these are the highest volume, the most resource intensive, the most functionally critical, or generate the most revenue. These transactions should be viewed from a business value perspective and should be selected to meet the stated purposes for testing.

After you have identified the key transactions, you will need to determine peak transaction volumes and usage patterns. You can often identify usage patterns by tracking historical CPU usage, or by processing the access logs of a similar application which is already in production. However, if it is a new application you may need to conduct more interviews to understand anticipated user behaviors and forecasted transaction volumes. If historical information is not available you should be conservative in your testing efforts and round up their estimates.

Be sure to keep in mind that very few applications have all enrolled users active on the system at the same time. Between people using the application in different time zones and different shift schedules usage will vary. In other words even if you have 1,000 enrolled users of the application being tested, most applications will have less than half of those users at peak, and sometimes as low as 10% concurrently active at peak time. If you size your system to handle 1,000 users and the peak only requires support for 100 users you will have wasted a great deal of money.

7. Proposed Test Scenarios (Template #2 – The Test Plan)

A set of tests should be designed for each designated purpose of the testing. Each test scenario should be run at least twice to confirm a consistent result, and if necessary re-run until the results are consistent. Results can vary from test to test depending on many variables that need to be controlled. Some examples of variability between tests include caching, database updates, and shared server resource conflicts.

In the case of caching the first time an application is tested perhaps nothing is cached, but each subsequent test the transactions could be cached. In the case of database updates the first time through a test the tables are updated, each subsequent test the data already exists potentially improving response times or resulting in errors. The common ways to avoid this scenario is have cyclical data (add a record, delete a record) or reset the database after each test. Resource conflicts with shared test environments occur when other applications not part of the load test consume resources in the middle of some tests, and other times there is no conflict. This can be avoided by scheduling dedicated use of the environment or not using a shared test environment. Ultimately your goal is predictability between tests and into production. Remember initially unpredictable results, errors or discrepancies, while not ideal, are often a chance to better understand the application and avoid more problems in production. Don't just ignore discrepancies as they are opportunities to learn more about the application.

Tests should be run for a sufficiently long time to get a statistically valid sample of activity. For example, if your script takes 10 seconds to run through once then perhaps 10 minutes would be sufficient resulting in 60 iterations per VU. Similarly, if the script only takes 100 milliseconds to run (like LDAP) then a 2 minute test could be sufficient resulting in 1200 iterations per VU. When determining how long to run the tests be sure to also consider the monitoring data

collection intervals. If you can only collect system data on 2 minute intervals with your tools then the test needs to run for at least 5 and even better 10 minutes to have enough representative data.

For Performance and Stress tests you will design tests to alter the amount of load by changing variables like the number of virtual users, and adjusting the think time or pacing. In my experience the # of virtual users will make the greatest amount of change followed by think-time, and finally pacing changes will have the most subtle impact.

For tuning tests you want to select a specified load to generate. Once established you will run the same test settings over and over again as you change one application or system variable at a time to compare the results. The load volume should be high enough that improvements or degradation will be identifiable. The notes or "other" variable in the next section can be used to track the tuning variables.

For availability load testing select a specified load to generate and run the same tests over and over again as you shutdown, crash, failover, and then restart various components. With the example of a load balanced web and application server application you would want to run a sequence like the following for performance comparisons.

- a. clean test with all systems running
- b. shutdown 1 web server -----and recover
- c. shutdown 1 web AND 1 app server -----and recover
- d. shutdown 1 app server -----and recover
- e. crash 1 web server -----and recover
- f. etc.
- g. re-validate environment to ensure that it is back to pre-availability testing status.

You can quickly see how many tests would need to be done if you select a large scope of analysis, and have to run all of these twice to check for consistency.

For stability load testing select a specified likely load to generate and run the same load settings over and over again as you extend the duration of the tests. It is unlikely that these tests would be done with your peak load, but rather a high percentage of peak. This load should be high enough to keep the system busy but not overly stressed, unless of course your expected peak will be for an extended duration perhaps like a batch job designed to use all available resources.

8. Load Generation Requirements (See Template #3 - Generating the Load worksheet)

There are many options when it comes to load generation tools. You may choose to use QA Load, LoadRunner, SLAMD or some other load generating tool depending on what you are trying to test. Most tools have two primary components 1) a controller or conductor which configures, starts, monitors, and reports on the tests, and 2) one or more systems that actually simulate the clients and generate the load. Sometimes these components run on the same machine though that is not always the best approach. It can cause the controller to contend with the load generator for resources which can invalidate your test. Controllers do not usually need to be very large, but the quantity and size of load generators can range from one small server to many very large servers, depending on the load that will need to be generated.

When generating the load you will be using "virtual users" or VUs that will follow a script as if they are real users. You will need to decide how many VUs are necessary to meet your test purposes. Keep in mind that this does not have to be a one to one ratio of real users to virtual users because as a tester you can manipulate other variables like think time or pacing. In other words, you may only have a 500 VU license and your test is supposed to simulate 1000 real users. This can largely be overcome by simply reducing the think time in your script.

Another requirement is the proximity of the load generators to the test servers. Are they in the same datacenter on the same LAN or are they in geographically disparate locations? Tests conducted on a LAN or over a frame relay will be faster than tests conducted across a WAN. Ideally the load generators should be in a location that mimics real user localities on the network.

9. Proposed monitoring, tracking and reporting

You need to track enough data that you will be able to recreate the test and the results if necessary. You should also have enough information that you are able to interpret how much load you created, and the performance characteristics of that load. You may choose to use several different tools to track the details of resource utilization,

response times, and test settings, but tracking a summary of all this information in a single location will allow rapid comparison of results between tests.

For all tests be sure to track the test number, for reference, as well as the start and end dates and times so that you will be able to sync up results from the various tools.

The specific tools used, and the resulting data available will control some of what is tracked, but all tools should include the following or equivalent metrics.

Test Identifiers and Scenario Settings/Control Variables

This should include every configuration necessary to recreate the test and will vary depending on your tool(s) of choice.

Test type (Validation, Tuning, etc.), transaction mix, # Virtual Users (VU's), ramp options (VU steps and timings), delay between transaction executions (think or sleep time), test duration, logging level, caching options, any tuning parameters, etc.

Table 1: Sample Test Identifiers

Test Identifiers					
Test #	Test Date	Test Start time	test duration (min.)	Test Purposes	Notes
1				Performance	
2					
3					

Table 2: Sample Scenario Settings/Control Variables

Scenario Settings/Control Variables							
# of Iterations	Logging	Cache	Ramp Up	Pacing	Think Time	#Virtual users	Other (tuning)
	Standard	None	0	0	5	50	
	Low Level	Client	0	0	10	1	
	None	Server	0	0	0	1000	

Results

These are the relevant metrics that were measured in regards to response times, transaction rates, and resource utilizations, which will be used to write the report and prove that the various objectives were met. There are often metrics that are gathered to rule out certain limitations, but will not be tracked for every test. For example, once network utilizations have proven to be insignificant they no longer need to be tracked in detail.

Every tester has specific metrics they value, but the following generic list will account for the basics. Depending on the specifics of your test some of these won't be necessary, and other times you will need all of these and more.

Transaction Rates achieved for each transaction in each script or use case, and their corresponding response times (average, maximum, 90%, minimum) should be tracked in detail. For complex testing scenarios that use many transactions in many scripts the results tracking and documentation can get quite involved.

Table 3: Sample Response Time Results

Total Script Run Time	01 - Hit Server	02 - Login	Transaction 10	99 - Logout

Avg	Max	90%	Avg	Max	90%	Avg	Max	90%	Avg	Max	90%	Avg	Max	90%

The typical system metrics should also be tracked for each of the systems involved in the test, including the load generators (until they are identified as insignificant). For each system/server involved in the testing the system metrics that are recommended to start with are: CPU (avg/peak), Memory (avg/peak), Paging (avg/peak), Disk IO (avg/peak).

Table 4: Sample Server Metrics Results

<Server 1> - <role>			
CPU % (peak/ avg)	CPU Q length (peak/ avg)	Memory paging	Disk

You will also want to have a section for comments, notes, and issues. In this field you can list if there were any problems with the test or any observations that you made. It is also a good place to identify what variable (if any) was changed for that particular test. Many different tests and iterations of tests will likely be done and this level of documentation will help ensure that you don't have to re-run tests because you do not know what happened the last time it was run.

Conclusions

As you can see it takes many different skills and activities that usually cross organizational groups to perform a good load test. Hopefully you also see that collaboration between these groups provides checks and balances that further increase the quality of the testing and the subsequent decisions based on these efforts. With proper planning, and consistent communication you can conduct meaningful load tests.

Template #1 – Transactions

- Meeting Topics: Current Transaction Mix
Usage Pattern / Peak usage
- Attendees: Load Tester, Application/Business Representative, Capacity Planner(optional)
- Timing of Meeting: Before 1st Test (1 hr.)
- Agenda:
 1. Review the data that has been gathered so far and identify any gaps
 2. Review the current usage patterns to identify peaks, or update them if necessary
 3. Identify the transaction mix at peak
 4. Identify how projected growth will affect transaction mix

Usage Type	Day(s) of Week	Start Time	End Time	Transaction Types
Interactive - Normal				
Interactive - Peak				
Batch - Normal				
Batch - Peak				

What

Business Activity Name	Description	Peak Day(s)/Time	Peak Volumes	Affected Systems
A				
B				
C				

Details - How

Primary Transaction	Transaction	Measured Response Time (Y/N)	Response Time Requirement (Seconds)	Notes
Business Activity A	#1 – Login			
	#2 – Click XX			
	#3 – Click YY			
	#99 – Logout			
Business Activity B	#1 – Login			
	#12 – Click XX			
	#13 – Click YY			
	#99 – Logout			

Template #2 – The Test Plan

Meeting Topic: Test Plan Creation
Attendees: Test Manager, Load Tester, Infrastructure, Capacity Planner.
Timing of Meeting: Before 1st Test (90 min.)
Agenda:

1. Develop specific test scenarios to accomplish the purposes of the testing
 - a. One or more of the six possible
 - b. Based on peak loads and transaction mixes
2. Ensure the test environment is properly planned and scheduled

Remember:

- Test duration should allow for many transactions
 - If a transaction takes 300 milliseconds to complete, then 5 minutes is a long time
 - If a transaction takes 2 minutes to complete, then 20 minutes may not be long enough
- As tests are run some adjustments may need to be made in Virtual Users, Think Time, or Pacing in order to get the right transaction mix

Script Name	Measured Transactions	Scenario % of Whole	# of Virtual Users	Think Time On/Off	Pacing	Typical Test Duration	Stability Test Duration
A	Trans #1	50					
	Trans #2						
	Trans #3						
	Trans #4						
	Trans #5						
	Trans #99						
B	Trans #1	30					
	Trans #8						
	Trans #99						
C	Trans #1	20					
	Trans #99						

Template #3 – Generating the Load for the Test

Meeting Topic: Load Generator identification/installation
Attendees: Load Tester, Infrastructure, Capacity Planner (optional)
Timing of Meeting: Before 1st Test (30 min.)
Agenda:

1. Identify specific servers to use
2. Determine who will do the install of the load generator
3. Ensure we have all the code needed to do the install
4. Identify who will run the tests

Load Generating Tool:

- QA Load
- LoadRunner
- eTester
- SLAMD
- Other – Please Specify

Load Generation Machine(s)	VU's or Transactions Per Machine	Installed and Ready (Y/N)	On Appropriate Physical Network (Y/N)	Network Bandwidth Simulation (Y/N)	Monitoring of Load Gen Machine(s) in Place (Y/N)

Template #4 – Initial/Validation Tests

Meeting Topic: Initial (low level) load tests
 Attendees: Load Tester, Application/Business Representative, Infrastructure
 Timing of Meeting: When Scheduled for Exclusive Access (varies)
 Agenda:

1. Verify that the test environment is properly set up, configured and functioning as expected
2. Perform Initial load tests
3. Validate test plan compliance with the load tool
4. Provide preliminary results to the team

Things to Keep in Mind:

- Did all of the scenarios and transactions run properly?
 - Did the UserIDs work?
 - Did the passwords work?
 - Was appropriate user data available?
- Were response times for one or two users acceptable and understandable?
- Did the observed resource utilization make sense?
 - Load generation machines?
 - Application servers?
- Note any issues with scripts:
 - Execution errors
 - Failed logins
 - Timeout errors
- Note any General Concerns:
 - Did you have exclusive access to the target servers?
 - Were there one or more unknown workloads using CPU when your testing was idle?
 - Was a backup running?
 - Where there unrelated batch jobs running?

Use Test Log tracking sheet

Template #5 – Initial/Validation Tests – Report (Optional if Validation Testing Had No Issues)

Meeting Topic: Initial load tests Review/Report
 Attendees: Test Manager, Load Tester, Application/Business Representative, Capacity Planner

Timing of Meeting: Week after Initial load tests (1 hr.)

Agenda:

1. Review initial load tests for any major changes needed
2. Schedule major load tests

Template #6 – Primary Load/Performance Tests

Meeting Topic: Major load tests (find the application limits, fill the purposes of the testing)

Attendees: Test Manager, Load Tester, Application/Business Representative, Infrastructure
Capacity Planner

Timing of Meeting: Days after Initial load tests (varies)

Agenda:

1. Baseline each script
2. Finalize scenario settings to get required transaction mix
3. Identify/correct bottlenecks, if any (player machines, web servers, app servers, db servers, etc.)
4. Once testing is functioning as expected, perform major load tests including coordination with Network team.

Template #7 – Primary Load/Performance Tests - Report

Meeting Topic: Major load tests Report

Attendees: Test Manager, Load Tester, Application/Business Representative, Infrastructure
Capacity Planner

Timing of Meeting: Week after Major load tests (1 hr.)

Agenda:

1. Present to the team the results of the testing
2. Determine if results are satisfactory or if additional testing is required.