Avoiding Performance Downtime and System Outages Through Goals-based Testing

Nate Skinner, product manager, Embarcadero Extreme Test
and
Paul Down, senior consultant, Embarcadero Technologies, Ltd.

August 2003

Corporate Headquarters
Embarcadero Technologies
425 Market St., Suite 425
San Francisco, CA 94105
www.embarcadero.com

EMEA Headquarters
Embarcadero Technologies, Ltd.
Thames House
17 Marlow Road
Maidenhead
Berkshire
SL6 7AA
United Kingdom
www.embarcadero.co.uk

Table of Contents

Avoiding Performance Downtime and System Outages Through Goals-based Testing

August 2003

Introduction

Database, application, and system performance and availability have become critical factors directly affecting a company's bottom line. Enterprises are facing increased pressure to compete and deliver enhanced performance for business processes, which is intensified by the growth in data and the increased complexity of application architectures. IT organizations are also seeing an increase in the number of applications that they are required to manage, tighter budgets, and demands for increasing levels of customer service.

How does a company remain competitive and profitable in today's challenging marketplace? One answer is through performance testing of their enterprise systems. Performance testing refers to the process of systematically evaluating the performance of applications under stress to identify potential bottlenecks and to assess scalability. The idea is to predict the performance of an application and to fix bottlenecks before costly problems occur in production.

Testing applications using traditional methods is a time-consuming, error-prone process that places a heavy burden on the performance engineer who is tasked with delivering accurate results, quickly. Ideally, performance testing should mimic production activity — including users, activities, and data — under test conditions in which the amount of load applied against the application can be varied. Traditional testing methods do not offer a streamlined way generate accurate results with a granular level of detail in a short amount of time. A goals-based approach to performance testing dramatically accelerates and streamlines the testing process, 6*making it easier for performance engineers to do their job.

This paper examines goals-based testing as a new approach to performance testing of enterprise systems. It highlights the challenges facing today's businesses and demonstrates how a goals-based approach to performance testing helps. In addition, it presents a goals-based performance testing solution available from Embarcadero Technologies.

What Is Goals-based Testing?

Goals-based testing gives users the ability to ask critical performance questions before a test begins. This approach to testing promotes a proactive methodology by enabling test professionals to set thresholds and parameters on the test to ensure that the outcome meets all defined requirements. This ability dramatically accelerates the testing process as tests can be aborted once thresholds are exceeded. Further, testing is of little use unless user activity under production-like circumstances can be accurately reproduced. Simulating realistic user activity can be quite complex due to the variety of variables involved. The traditional approach to testing is to fire off a test on the system and hope that it can handle the load and meet business specifications. A goals-based approach allows users to set criteria for success (or failure) in terms of thresholds and realistic user scenarios before the test begins.

When goals are configured in the performance test, the test will answer the business question of "how many users" much more directly, run tests in less time, and require less analysis after the test run. A benefit of goals-based performance testing is that the performance test tool does the work of evaluating the current performance of the application infrastructure and the performance test infrastructure.

With a goals-based approach, users identify the goals of the test, and then set the criteria accordingly — defining acceptable thresholds for the various parts of the system (response times, CPU utilization, etc.). Upon execution of the test, the testing tool will exit the test if any of the defined thresholds are exceeded. This approach removes the typical try/repeat cycle that is so common with most load-testing processes. Once a test begins with these criteria set, if any of them are met (or exceeded), the test professionals save significant amount of time they would otherwise have spent waiting for a standard test to finish, only to discover that the performance levels were unacceptable partway through the test.

How Does Goals-based Testing Work?

Goals-based performance testing promotes a proactive testing methodology in which performance thresholds are defined in the beginning of the performance test process and configured in the performance test itself. Taking a non-goals-based approach — the traditional testing approach — to testing leaves defining performance thresholds until after tests are run, and sometimes not at all. Generally, the goals of performance testing are to determine how many concurrent users your system can handle while continuing to deliver a positive user experience and to ensure batch jobs finish within reasonable times, etc. To meet these goal, the test must answer some important questions:

1. What are acceptable user performance levels?
2. How many concurrent users are required?
3. How fast should the site respond?
4. What are acceptable levels of system utilization? (CPU, memory, network, disk, etc.)

Traditional performance testing makes it difficult to answer these questions with the level of detail businesses require to remain competitive. All of these questions can be answered at a general level, but to reach a more granular level, users need the ability to specify goals and to program these goals into the performance test itself. Using traditional testing methods, users configure tests by guessing the number of users the system can support. If they guess wrong, they must configure and execute another test. This process is repeated over and over until the four questions above are answered. If the users are lucky, they meet their goals in the first few test executions. If they are not, it could take many iterations before the test achieves its goals.

Goals-based testing allows users to define the performance goals for the application before the test begins and feel more confident that the results returned are accurate. The goals-based approach makes it possible for the user to answer the question: How many users can my application infrastructure support with *X* second transaction response times and without risk of system overload due to lack of available resources?

The Two Sides of Performance Goals

The Stakeholders

There are distinct stakeholders in the input and output of performance testing, and each has unique concerns when it comes to the goals of the test:

1. Business sponsor of the application — primary concern is end-user experience
2. Technical team (the application development team, the application infrastructure group, and the QA group) — primary concern is system performance and availability.

The business sponsor specifies performance parameters from the end-user's perspective. While the ultimate goal is fast Web page and results delivery to end users, the performance parameters cover both transaction response time as well as the number of users the system must support. And, while the business sponsor may take a variety of approaches to determine the values for the performance test parameters — historical information and known data is one approach — the bottom line is always "if the system does not support a certain number of users we won't be profitable." This is where the technical team enters.

The technical team is focused solely on system performance. System performance is the foundation for a positive end-user experience. System performance refers to acceptable levels of system resource utilization under stress. Since performance testing is the process of applying load to the system to determine how it reacts, these parameters are very important to include in the performance testing process.

In production, businesses do not want to see their CPU utilization at 100 percent for extended periods of time because this generally means that the end-user experience is substandard. And, at 100 percent utilization, it is a good bet that any increase in the number of concurrent users is going makes the end-user experience much worse. Nevertheless, thresholds pertaining to CPU utilization are often not specified until later in the performance testing process which leaves both sides — the business sponsor and the technical team —wondering if it is really acceptable that the system meets business requirements at 96 percent CPU utilization. By specifying these thresholds early in the process, the performance of the system is clearly understood to be either acceptable or not acceptable.

It is up to the technical team to ensure that systems are always optimized, running at peak levels, and that resources are utilized effectively. This ultimately results in a positive end-user experience, which satisfies the business sponsor's concern. The business sponsor and the technical team have to work together to create an organized performance testing process and meet the bottom line of the business — profitability.

Ultimately, the business sponsor and the technical team must find a way to achieve positive end-user experience without taxing the infrastructure. This places a heavy burden on the performance-test engineers. They are tasked with determining if the application infrastructure and the performance test infrastructure are performing well, which is challenging under normal circumstances, and can become a major headache when there are many performance thresholds, including many different transaction response times and system performance statistics. Add to this the fact that IT organizations expected to manage an increasing number of applications and meet demands for increasing levels of customer service with tighter budgets and less staff.

Traditional Performance Testing and the Two Sides of Performance Goals

Almost every application suffers from some performance problems. It is rare to find a well-tuned application and even harder to find one that stays that way in the face of growing user loads, network traffic, and data volumes. Neglecting performance problems can lead not only to poor end-user experience, but even application outages.

The goal of a performance test is to determine how the system responds under stress, and, more important, how it can scale. Additionally, it is important to find out whether or not resources are being utilized effectively. With most performance testing processes and the tools that support them, users start a test with no idea what the system's performance capabilities are above a certain amount of load. Their system performs acceptably at 100 users, or 500 users, but can it handle 1000 users? Can it handle 5000? If the system failed to handle a larger load, what was the reason? Available system resources? A bottleneck in the database? If so, at what point did the database begin to bottleneck — was it 575 users, 600? A combination of these factors? Or something altogether different. The possibilities are limitless.

For example, let's look at an Internet company whose business is growing at a quick pace. To meet the new service level demands, it purchases new systems, which need to be tested before they are deployed into the production environment. A business plan is created, and it specifies that the new systems must be able to handle 1000 concurrent users and deliver pages in eight seconds or less, 90 percent of the time. On the other side, the technical team tasked with the implementation want to keep system CPU utilization to 85 percent or less. Each has a different goal, but both are tasked with satisfying the bottom line — application availability and a positive end-user experience.

The first step in the process requires an initial test to get a sense of system performance — to determine the breakpoint. To do so, the performance engineer must create a time-based test that essentially ramps up the load until the system fails, meets the target, or exceeds it. Note: since a changing load stresses the system differently than a constant load, it's a good idea to test the system with a constant load at the breakpoint.

In the first test, the eight-second response time was exceeded at 700 users. So, the initial breakpoint was determined to be 700 users. Also, CPU utilization was at 90 percent. However, the performance engineer did not know this until the test completed. Also, response time was measured at eight seconds, which was the transaction goal, but supporting 700 users was not acceptable based on the business plan, and neither was 90 percent CPU utilization per the technical team's requirement.

The performance engineer then makes an application code change with the hope of improving performance, and then executes the test again. The next iteration returns a breakpoint of 775 users with an eight-second response time, with CPU utilization under 50 percent. This was a small improvement in the number of users, which did not make sense given the drop in CPU utilization, but it still did not meet the test goal of 1000 concurrent users, eight-second transaction response time, and 85 percent CPU utilization.

The performance engineer continues this time-consuming, iterative process of creating and executing tests, watching various system counters to determine where the system exceeds thresholds or fails to meet requirements. After several test runs, the technical team determines that the database running behind the Web application was the bottleneck. The team reimplements a stored procedure, and then executes the break-point test once more. They ramp from 0 users to 1000 users over time. Upon completion, the test results indicate a response time of 6.5 seconds at 1000 users, and CPU utilization at 60 percent. The test met the business requirements.

At this point, the performance engineer stopped because he met the requirements — 1000 concurrent users, less than eight-second response time, and he did this using less than 85 percent of resource utilization. However, what still remains a mystery is how many users could the system handle before it exceeded the thresholds. To determine this, the performance engineer would have to continue the testing process. Also, because CPU utilization was lower than the requirements, if the business and/or application infrastructure team wanted to look at resource utilization, another test must be configured and executed, possibly multiple times.

The fact is that traditional testing is a trial and error approach, and does not return the critical information regarding how many users the system can handle. Also, performance engineers must wait until the test completes to determine where the system failed. Once they have made this determination, the next step is to tune the system and rerun the tests. After several of these cycles, a good performance engineer can determine what changes are required to allow the system to scale to the level required by the business. However, there are several problems that can cause even the best performance engineer to be forced to execute numerous test runs. And, when the performance engineer runs the test a number of times, he may still never find out how many users the system can really handle.

Traditional testing methods make it harder, not easier, to meet increasing business demands. Ultimately, the business suffers because it either has to accept results that do not provide the level of detail required to make sure all objectives are meet as efficiently as possible, or it has to increase staff to gain the insight required. This creates a conflict between the business sponsor and technical team. Each is relying on the other to meet the bottom line — profitability — but neither can afford to forgo their unique objectives. In the end, often compromises are made that reduce the confidence either team has in the test results, and potentially expose the company to performance downtime and, worse, a system outage.

How Goals-based Testing Is a Better Approach

Goals-based testing alleviates the conflict between the business sponsor and technical team. It provides the means for asking questions at the level of detail businesses require to remain competitive. And goals-based testing makes it possible for the technical team to determine the exact requirements for the system from the outset, again answering the four critical questions:

1. What are acceptable user performance levels?
2. How many concurrent users are required?
3. How fast should the site respond?
4. What are acceptable levels of system utilization? (CPU, memory, network, disk, etc.)

Goals-based testing allows the performance engineer to define the performance goals for the application before the test begins. This approach makes it possible to ask the questions: How many users can my application infrastructure support with *X* second transaction response times without risk of system overload due to lack of available resources? How can I feel confident that the results returned are accurate?

A goals-based test includes a series of phases and exit conditions. Within a phase, if the test exceeds any of the designated thresholds, it carries out the designated exit conditions. Also, users can set thresholds that will simply stop the test if exceeded. The user can define and configure the test with all thresholds specified. Knowing that the goals-based performance test takes advantage of these thresholds, the user can proactively identify them and specify their value.

In the previous example, numerous iterations of the test were executed. An initial test determined that the system could support 700 users with an eight-second-response time, and that CPU Utilization was at 90 percent. The second determined that the system could support 775 users with and eight-second-response time, but the CPU utilization was low compared to the response time. After this test, the technical teams determined that the database system running behind the Web application was the bottleneck, which explained the reason for low CPU utilization on the application server and the long response times — the application server was waiting around for the database to respond. The technical team corrected the problem and executed the test achieving 1000 users with response times under 8 seconds and CPU utilization at less than 85 percent.

Using a goals-based approach in this scenario, the test would look very different. The performance engineer would identify the test criteria, and then create a test with specified thresholds based on that criteria.

The Test Criteria

- 1000 concurrent users
- 8-second response time 90 percent of the time (90th percentile)

- Less than 85 CPU utilization
- Monitor complete application infrastructure (database, Web server, application server, etc.)

The Test

Phase 1 – Ramp Up until we exceed eight second response time, or exceed 85 percent CPU utilization.
If test reaches eight seconds with fewer than 1000 users, exit the test.
If CPU utilization exceeds 85 percent, exit the test
If the test exceeds eight seconds and concurrent users are more than 1000 users, exit the phase.
Phase 2 – Test the system at a constant load.
If CPU utilization exceeds 85 percent, exit the test
When the phase runs for 15 minutes, exit the phase.
Phase 3 – Ramp down until there are no more users

The Results

The performance engineer ran the test and it exited at Phase 1 because CPU utilization exceeded 85 percent. This was determined quickly since the engineer did not need to wait until the test fully executed. In addition, because the performance engineer set monitoring the database and application server as one of the test parameters, he also immediately identified the bottleneck in the database layer and corrected it. The next test ran to completion. The returned results were the same as in the previous example with a great deal more detail. Specifically, the goals-based test indicated the page delivery time for 1000 concurrent users, and it indicated the number of concurrent users required to result in pages being delivered at a rate of eight seconds.

By taking a goals-based approach to this scenario, the performance-test engineer was able to accurately answer the question “How many concurrent user can the system support with acceptable performance?” Additionally, the goals-based approach minimized the time required to adjust and reconfigure time-based performance tests to answer the same question. For example, in a time-based performance test, a test schedule is configured. In a goals-based performance test, a load model is configured and thresholds are defined so that instead of a applying a time-based transition from increasing users to a constant user load, the ramp continues until a threshold is exceeded. The result is either the thresholds are exceeded and the phase ends and the next phase starts, or, the whole test terminates because performance is unacceptable to a point that there is no reason to continue executing.

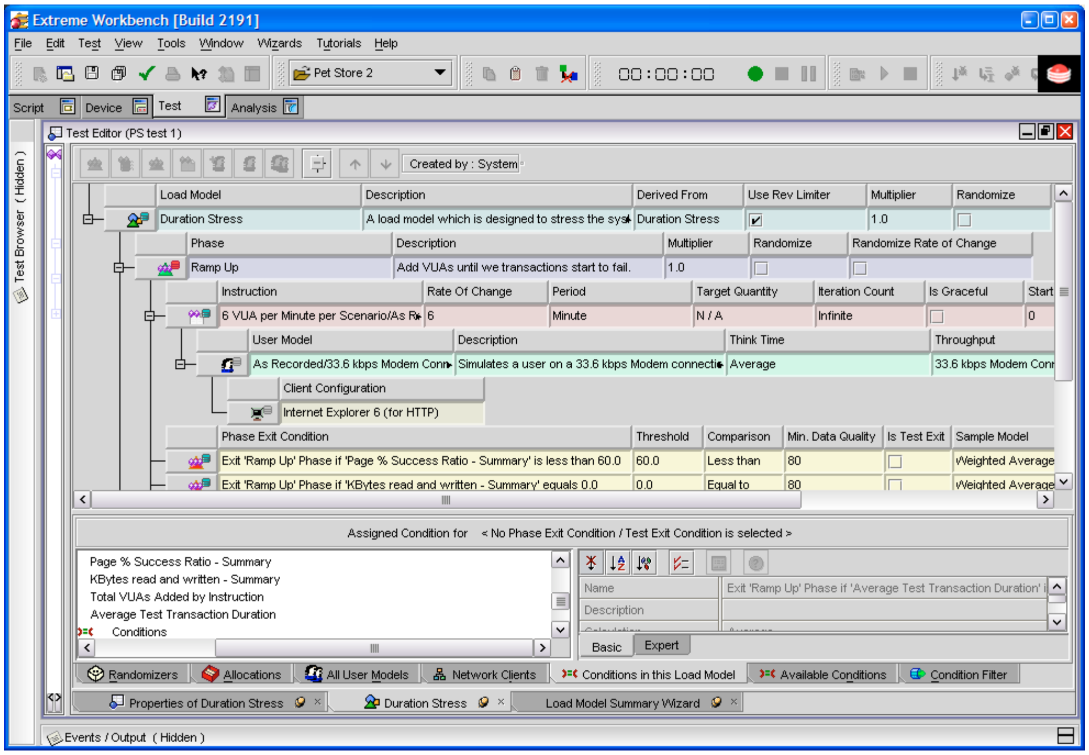
A Goals-based Testing Approach with Embarcadero Extreme Test

Extreme Test is designed to embrace a goals-based approach to performance testing. Tests created with Extreme Test adapt to users goals — all of them. Like other load testing tools, performance engineers can specify a number of users as the testing criteria. And Extreme Test can capably monitor the software under the stress when it deploys these virtual users. But Extreme Test is better adapted to create advanced load models that also take into account system thresholds.

An Extreme Test Load Model is constructed of Phases, allowing users to accurately mimic increases, decreases, spikes, and lulls in user activity. Each Phase is itself broken into

Instructions, allowing user to create complex Phases. Each Instruction can define a different user simulation, meaning that Instructions dictate what types of users the Extreme Test Engines generate.

For example, an Instruction might dictate that its users have a slow think time — that they take a long time between clicks — but still have a high bandwidth. Furthermore, the Instruction might dictate that many of them use Internet Explorer. A second Instruction could have the opposite: Fast-thinking users with low bandwidth and Netscape. Each Instruction in a Phase runs in parallel, making the Load Model's Phases incredibly lifelike.


Extreme Workbench [Build 2191]
File Edit Test View Tools Window Wizards Tutorials Help
Pet Store 2
00:00:00
Script Device Test Analysis
Test Browser (Hidden)
Test Editor (PS test 1)
Created by : System

Load Model	Description	Derived From	Use Rev Limiter	Multiplier	Randomize
Duration Stress	A load model which is designed to stress the sys	Duration Stress	✓	1.0	☐

Phase	Description	Multiplier	Randomize	Randomize Rate of Change
Ramp Up	Add VUAs until we transactions start to fail.	1.0	☐	☐

Instruction	Rate Of Change	Period	Target Quantity	Iteration Count	Is Graceful	Start
6 VUA per Minute per Scenario/As R	6	Minute	N / A	Infinite	☐	0

User Model	Description	Think Time	Throughput
As Recorded/33.6 kbps Modem Conn	Simulates a user on a 33.6 kbps Modem connecti	Average	33.6 kbps Modem Conn

Client Configuration
Internet Explorer 6 (for HTTP)

Phase Exit Condition	Threshold	Comparison	Min. Data Quality	Is Test Exit	Sample Model
Exit 'Ramp Up' Phase if 'Page % Success Ratio - Summary' is less than 60.0	60.0	Less than	80	☐	Weighted Average
Exit 'Ramp Up' Phase if 'KBytes read and written - Summary' equals 0.0	0.0	Equal to	80	☐	Weighted Average

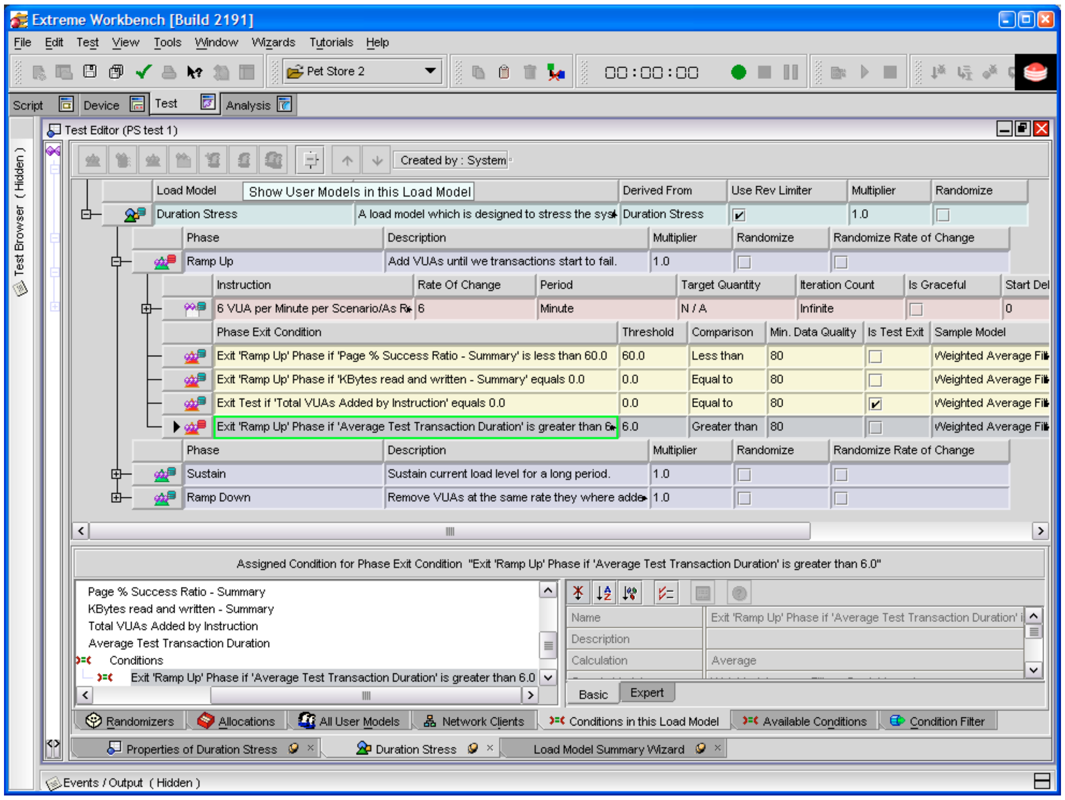
Assigned Condition for < No Phase Exit Condition / Test Exit Condition is selected >

Page % Success Ratio - Summary
KBytes read and written - Summary
Total VUAs Added by Instruction
Average Test Transaction Duration
Conditions

Name	Exit 'Ramp Up' Phase if 'Average Test Transaction Duration' i
Description	

Basic Expert
Randomizers Allocations All User Models Network Clients Conditions in this Load Model Available Conditions Condition Filter
Properties of Duration Stress Duration Stress Load Model Summary Wizard
Events / Output (Hidden)

Instructions bring depth to the virtual users that the Extreme Test Engines generate. Meanwhile, the Extreme Test Engines themselves are always aware of the system statistics that users deem critical to watch. Extreme Test's Load Models give users access to thousands of performance metrics across hundreds of software types and platforms. Users can establish any one of these performance metrics as a guideline that is honored during Test execution. For example, let's say that performance engineers determine that CPU utilization should never be more than 85 percent during the time period in which the system accommodates the desired amount of users. In a Load Model Phase, they can simply add this rule. Users indicate in the Load Model that if CPU utilization ever exceeds 85 percent, then the test has failed and should stop. The Extreme Test Engines, while generating Virtual Users, are always evaluating the system statistics returned by Extreme Test's monitors. And if those statistics exceed any thresholds that performance engineers have set, then the Extreme Test Engines can either exit the Phase or the Test itself, saving users the headache of having to wait an extended amount of time for an already-failed test to complete.


Extreme Workbench [Build 2191]
File Edit Test View Tools Window Wizards Tutorials Help
Pet Store 2
00:00:00
Script Device Test Analysis
Test Browser (Hidden)
Test Editor (PS test 1)
Created by : System
Show User Models in this Load Model
Randomizers Allocations All User Models Network Clients Conditions in this Load Model Available Conditions Condition Filter
Properties of Duration Stress Duration Stress Load Model Summary Wizard
Events / Output (Hidden)
Basic Expert

Load Model		Derived From	Use Rev Limiter	Multiplier	Randomize
Duration Stress	A load model which is designed to stress the sys	Duration Stress	✓	1.0	☐

Phase	Description	Multiplier	Randomize	Randomize Rate of Change
Ramp Up	Add VUAs until we transactions start to fail.	1.0	☐	☐

Instruction	Rate Of Change	Period	Target Quantity	Iteration Count	Is Graceful	Start Del
6 VUA per Minute per Scenario/As R	6	Minute	N / A	Infinite	☐	0

Phase Exit Condition	Threshold	Comparison	Min. Data Quality	Is Test Exit	Sample Model
Exit 'Ramp Up' Phase if 'Page % Success Ratio - Summary' is less than 60.0	60.0	Less than	80	☐	Weighted Average Fil
Exit 'Ramp Up' Phase if 'KBytes read and written - Summary' equals 0.0	0.0	Equal to	80	☐	Weighted Average Fil
Exit Test if 'Total VUAs Added by Instruction' equals 0.0	0.0	Equal to	80	✓	Weighted Average Fil
Exit 'Ramp Up' Phase if 'Average Test Transaction Duration' is greater than 6	6.0	Greater than	80	☐	Weighted Average Fil

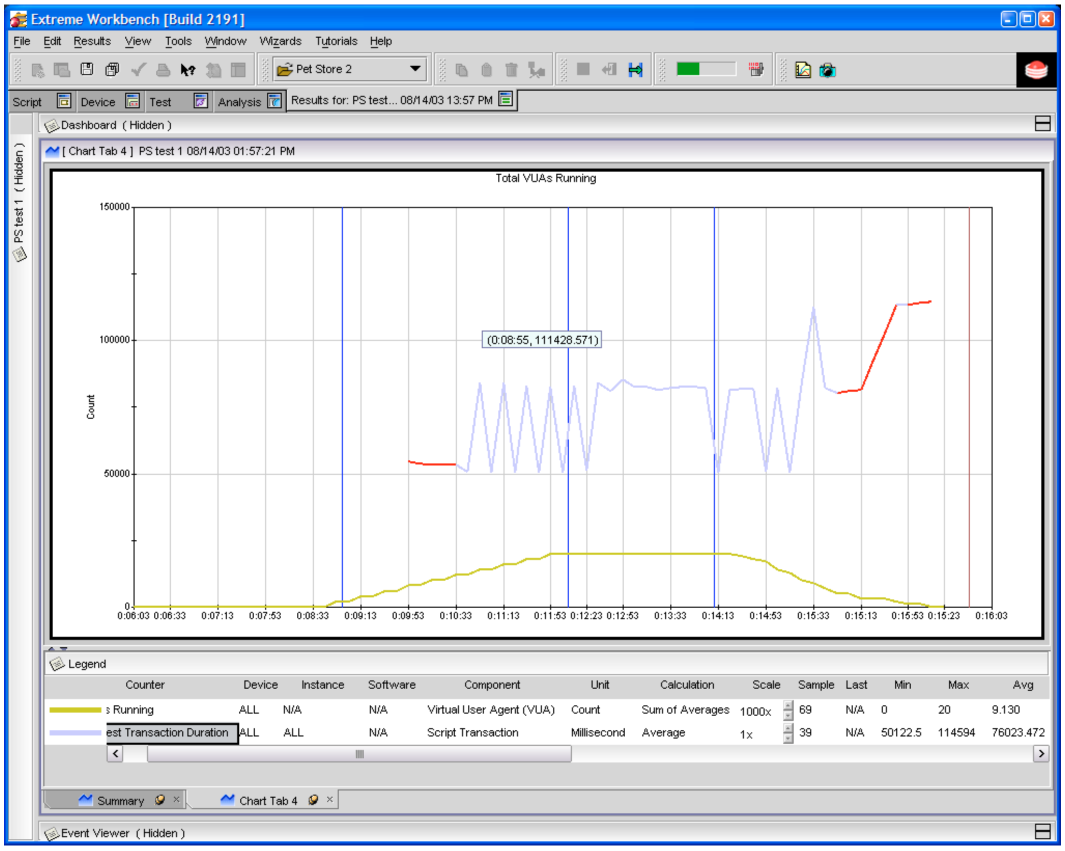
Phase	Description	Multiplier	Randomize	Randomize Rate of Change
Sustain	Sustain current load level for a long period.	1.0	☐	☐
Ramp Down	Remove VUAs at the same rate they where adde	1.0	☐	☐

Assigned Condition for Phase Exit Condition "Exit 'Ramp Up' Phase if 'Average Test Transaction Duration' is greater than 6.0"

Page % Success Ratio - Summary
KBytes read and written - Summary
Total VUAs Added by Instruction
Average Test Transaction Duration
Conditions
Exit 'Ramp Up' Phase if 'Average Test Transaction Duration' is greater than 6.0

Name	Exit 'Ramp Up' Phase if 'Average Test Transaction Duration' i
Description	
Calculation	Average

When Extreme Test generates a report for this Test, the performance engineer need not sift through piles of data. S/he knows exactly where to look for the data that leads to any offending bottlenecks — at the end of the report, when the Extreme Test automatically stopped the Test. Extreme Test's real-time monitoring and reports can also key on system statistics that users may have identified, along with CPU utilization, as critical statistics. Users can configure Extreme Test's Workbench to displays these statistics during the test's execution and to highlight these statistics in post-execution reports. The performance engineer does not have to seek the most important information. The most important information is delivered to him in an easy-to-interpret format.



	Counter	Device	Instance	Software	Component	Unit	Calculation	Scale	Sample	Last	Min	Max	Avg
🟨	s Running	ALL	N/A	N/A	Virtual User Agent (VUA)	Count	Sum of Averages	1000x	69	N/A	0	20	9.130
🟦	est Transaction Duration	ALL	ALL	N/A	Script Transaction	Millisecond	Average	1x	39	N/A	50122.5	114594	76023.472

Summary

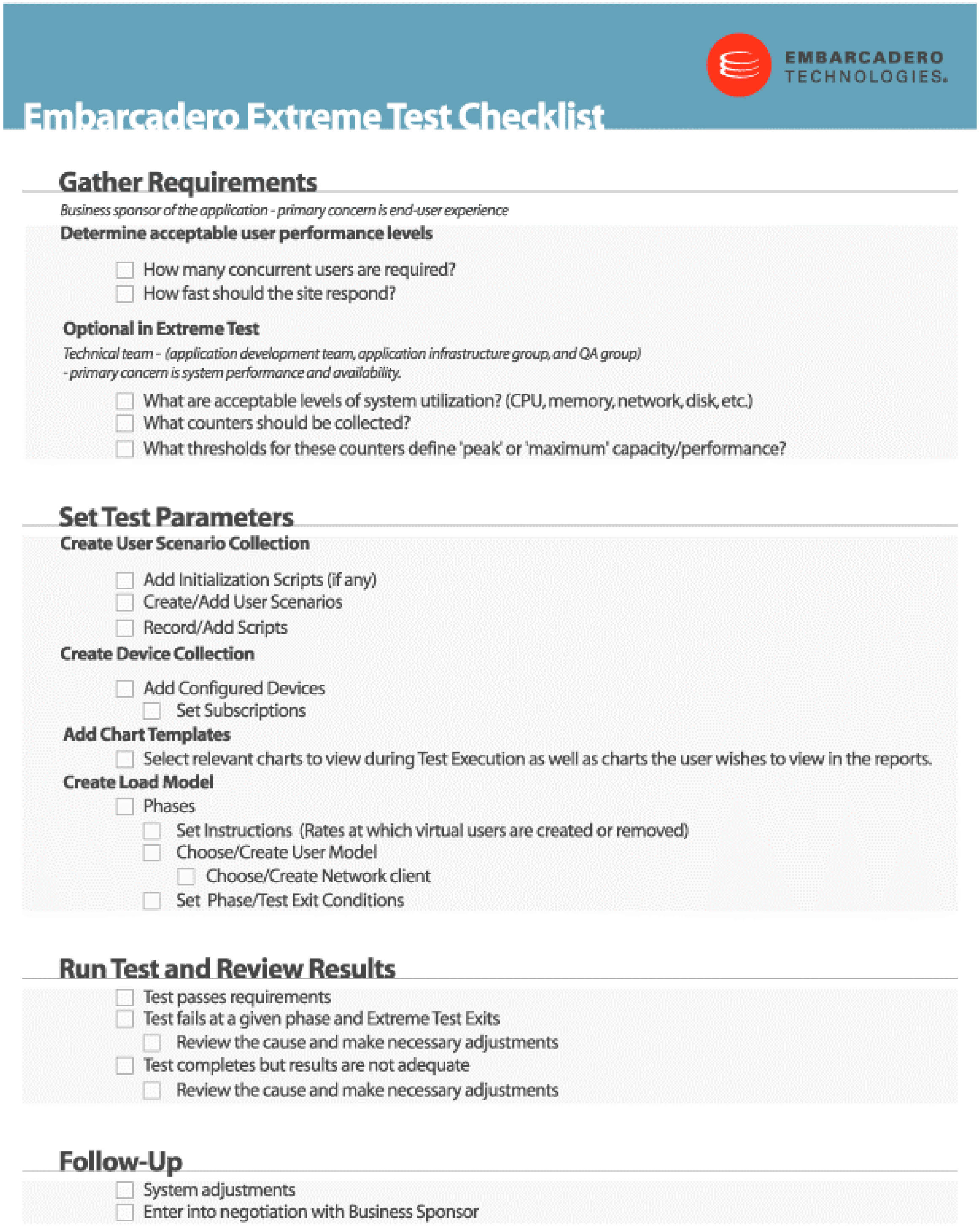
Almost every application suffers from some performance problems. It is rare to find a well-tuned application and even harder to find one that stays that way in the face of growing user loads, network traffic, and data volumes. Today's end user has zero tolerance for outages and slow performance. Explosive growth in technology means increasing system stress, and this leads to potential performance problems on ALL system components, including the database, Web server, and application. Neglecting performance problems can be costly. Poor performance leads to downtime, complete outages, and ultimately, a poor end-user experience. To avoid this, companies are adopting a broader performance assurance program in which the performance and stability of critical applications are regularly evaluated and optimized even after they reach production.

Embarcadero Extreme Test is a next-generation, goals-based performance testing solution. Its innovative approach to test creation includes timesaving features, unprecedented scalability, and open, standards-based architecture.

By using Embarcadero Extreme Test, IT professionals can have greater confidence regarding system performance when deploying and upgrading applications throughout the enterprise.

About Embarcadero Technologies

Embarcadero Technologies provides leading companies with enterprise software to manage and optimize their critical database and application infrastructures, allowing them to obtain better results with fewer resources. Embarcadero Technologies' software products enable companies to build, optimize, and better manage their critical business applications and underlying databases. Embarcadero leads the market in providing high-quality, easy-to-use database and application development tools that deliver cost-effective solutions.



EMBARCADERO TECHNOLOGIES.

Embarcadero Extreme Test Checklist

Gather Requirements

Business sponsor of the application - primary concern is end-user experience

Determine acceptable user performance levels

- ☐ How many concurrent users are required?
- ☐ How fast should the site respond?

Optional in Extreme Test

Technical team - (application development team, application infrastructure group, and QA group) - primary concern is system performance and availability.

- ☐ What are acceptable levels of system utilization? (CPU, memory, network, disk, etc.)
- ☐ What counters should be collected?
- ☐ What thresholds for these counters define 'peak' or 'maximum' capacity/performance?

Set Test Parameters

Create User Scenario Collection

- ☐ Add Initialization Scripts (if any)
- ☐ Create/Add User Scenarios
- ☐ Record/Add Scripts

Create Device Collection

- ☐ Add Configured Devices
  - ☐ Set Subscriptions

Add Chart Templates

- ☐ Select relevant charts to view during Test Execution as well as charts the user wishes to view in the reports.

Create Load Model

- ☐ Phases
  - ☐ Set Instructions (Rates at which virtual users are created or removed)
  - ☐ Choose/Create User Model
    - ☐ Choose/Create Network client
  - ☐ Set Phase/Test Exit Conditions

Run Test and Review Results

- ☐ Test passes requirements
- ☐ Test fails at a given phase and Extreme Test Exits
  - ☐ Review the cause and make necessary adjustments
- ☐ Test completes but results are not adequate
  - ☐ Review the cause and make necessary adjustments

Follow-Up

- ☐ System adjustments
- ☐ Enter into negotiation with Business Sponsor