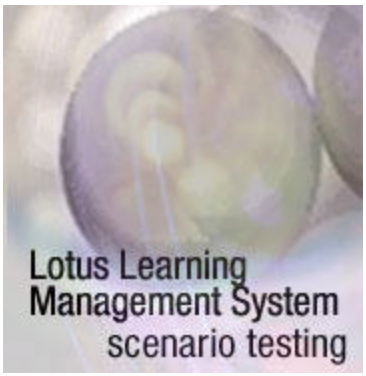


Level: Intermediate
Works with: Lotus Learning Management System
Updated: 15-Sep-2003

by
Linda
Martineau

Traditional software analysis usually involves feature testing, which puts a product through its paces one feature at a time. This can be an effective tool and is still widely employed by quality assurance teams. However, today's sophisticated software solutions often require that system testers place themselves in the shoes of the users to simulate real-world user experiences and activities. One way to meet this need is through *scenario testing.* This involves working with customers to determine a typical "day in the life" of a user and duplicating this as closely as possible within the test lab. Feature testing usually entails working with features individually. Scenario testing, on the other hand, helps us identify complex customer issues by taxing the system in a production-like manner. (This type of testing is also known as solution testing.)

This article explains how the Lotus Learning Management System (LMS) team developed scenario testing for LMS. We describe an example scenario and examine its components. This article assumes some familiarity with administering and testing LMS.

Scenario testing overview

To begin our scenario testing program, the LMS group took a step back from our usual feature test approach and looked at the entire product through the eyes of our users. We collected feedback and ideas from support analysts—those who are closest to our customers. We carefully listened to the details about the kinds of issues customers experienced with LMS to better understand how they used the product. This resulted in a list of the top 15 areas for us to focus on. We then used this list to build 20 unique scenarios for our team to run test cases against.

We tested our scenarios in a system environment that simulated actual user activity in a real customer production environment. To ensure our test closely matched the typical user experience, we ran multiple scenarios simultaneously on one complete platform. We also included different supported platforms from UNIX to Windows. Our goal was to validate that LMS performs well under complex, industrial-strength usage in an enterprise-level environment. We maintained roughly 700 test cases to support our actions and to manage our results.

Anatomy of a scenario test

Our scenarios consisted of the following components:

Handle
This is the name of the scenario, for example, Acme Worldwide Campus.

Sequence
This is a subset of activities within the scenario defined by the Handle.

Size
Also known as data population, this represents the size of the company (small, medium, or large), how many students are enrolled/not enrolled, how many courses are listed in the catalog, and so on.

Scope
This encompasses the geographical spread of the scenario (local, national, regional, or global).

Configuration
This lists the types of servers used during a given scenario test pass. This ensured that we tested all supported platforms at least once (but often multiple times during the LMS system test cycle). Problematic platforms were put through the test cycle until we identified and fixed all blocking issues, and the platform successfully passed our test.

Status
This reflects the status of each scenario: Active (ready to be run) or Draft (not yet ready for primetime).

Objective
This identifies what the scenario should accomplish, for example, setting up course content and customization. Unlike feature testing, this generally involves performing a number of tasks encompassing multiple features.

Background/Environment
This is the business model or type of business. For example, in one of our scenarios the Background/Environment section resembled the following:

This scenario will emulate Acme Worldwide Campus. This scenario will simulate a large company with many people all over the country, enrolling in and taking courses. We have duplicated typical day-to-day operations in test case form. This includes 200K Acme employees all over the country of which all 200K are rostered in Worldwide Campus and 180K are actively enrolled in at least 25 courses each. Peak times of the day are between 8 AM - 11 AM and 2 PM - 4 PM. Peak days occur when employees are required to take certain courses by a deadline, and the deadline approaches. There are 2,000 courses in the course catalog.

Precondition/Prerequisite
These are the system requirements for the test. These need to be stated up front for proper scheduling of the scenario to take place.

Data Set
This includes file attachments, file locations, links, URLs to files, LDAP directories, tools, and so on to populate the enterprise to start a given scenario. The following table shows a sample data set:

Total number of users in the LDAP directory	200,000
Number of LMS users who are Administrators (0.1 percent)	200
Number of LMS users who are Instructors (5 percent)	10,000
Number of LMS users who are Authors (5 percent)	10,000
Number of LMS users who are Students (90 percent)	180,000
Number of groups/profiles in directory structure (0.1 percent)	200
Number of courses in Course Catalog	2,000
Number of courses in which each student is enrolled on average	25
Expected number of concurrent users	2,000
Progress Records (assume 25 per user)	5,000,000

Roles/Tasks

In some respects, this is the "meat" of the scenario. This defines the types of users being simulated in the test and which tasks they perform. This needs to be sufficiently detailed to write individual test cases from it. Each task is transformed into a runable test case. For example:

Roles	Tasks
Course Author	1. Install the AAT on the client. 2. Asynchronously and manually create a course: - Create an assessment within the course. - Create a tracking-enabled live session within this course. - Export the course to the Content Management Server. 3. Import, modify, and export a Content Conversions third-party course. 4. Create an LVC activity, discussion database, and a tracking-enabled assessment in it. 5. After the course has been created, quit the job and uninstall the AAT. **NOTE:** Course authors or administrators should be able to take an active course off-line for editing. An automatic email notification should alert all students when the course is planned to go off-line and when it is back on-line. Automatic email notification should alert the students that the course is again available and where changes have been made to the course. These changes to the course should not affect any other courses or course sections that were created using the same course master or the course master itself. Also note that course master is the copy of the course that is used to create new sections of an active course.
Course Catalog Administrator	1. Import third-party vendor content: - NETg - SmartForce - SkillSoft 2. Register Course Masters for these courses. 3. Create course offerings for these courses, half self-paced, half classroom. 4. Assign instructors to courses. 5. Assign resources for courses and students. 6. Create and run a custom report from any place in LMS you choose (place link in custom location) to see how many Java courses are running next week.
Course Designer	1. Create a new customization set. 2. Edit that new customization set because a mistake was made. 3. Manage the users for customization.
Student	1. Launch QuickAnswers. 2. Perform a search. 3. Ensure that search results have returned for each Search Scope defined.

The Roles/Tasks form the central area of activity. By breaking dependencies down by sequence, we discovered it was easier for us to sift through the data to get the scenario to run in the right chronological order. This in turn helped define the roles and tasks that needed to be executed. This was important because if test cases were tagged in the wrong order, the tester might discover the suite of test cases did not run due to prerequisites from other actions. For example, if a test case took a specific course and the next test case enrolled in that course, the first test case was not executed because the user was not enrolled in the course yet.

Proper ordering of test cases was key to successfully executing a complete scenario. We grouped similar roles together in the order they needed to be executed, listing the steps relative to each role to accomplish the objective for the scenario. The roles in the scenario were independent of each other, which gave us some flexibility as to how many testers could run through a given scenario. Although this was an extra step in the process, we created a doclink to the test case that supported the task being defined. This gave us easy access to the test cases when test cases needed modifications from within the scenario form. We also had a number of views within the database that listed scenarios and all associated test cases, and whether they had a status of Pass or Fail. (This allowed our upper management to view status any time during the scenario test cycle.)

We had two different test types: functional (a test case written and executed by the function testers) and system (a test case written for scenarios, performance, metrics, and so on executed by our system test team). This eliminated duplication of effort and showed whether or not we were getting complete coverage of all features.

Scenario rules and requirements

After we had collected customer data and outlined our scenario structure, it was time to create some rules to guide us. This ensured that no matter who the author of the scenario was, anyone could run a given scenario because all scenarios were structured the same. We first decided that a scenario or sequence should take no more than a day to execute. This avoided running into too many dependencies on the previous sequence.

A scenario was not required to contain more than one sequence. A sequence could be based on a particular functional area when needed, but was not always limited to a functional area. Only one test case could be tagged to a task. Each task needed to be supported by a test case. These test cases in turn were linked to a metric for management to track and report against. We generated reports daily and cumulatively by week.

Reviewing a scenario

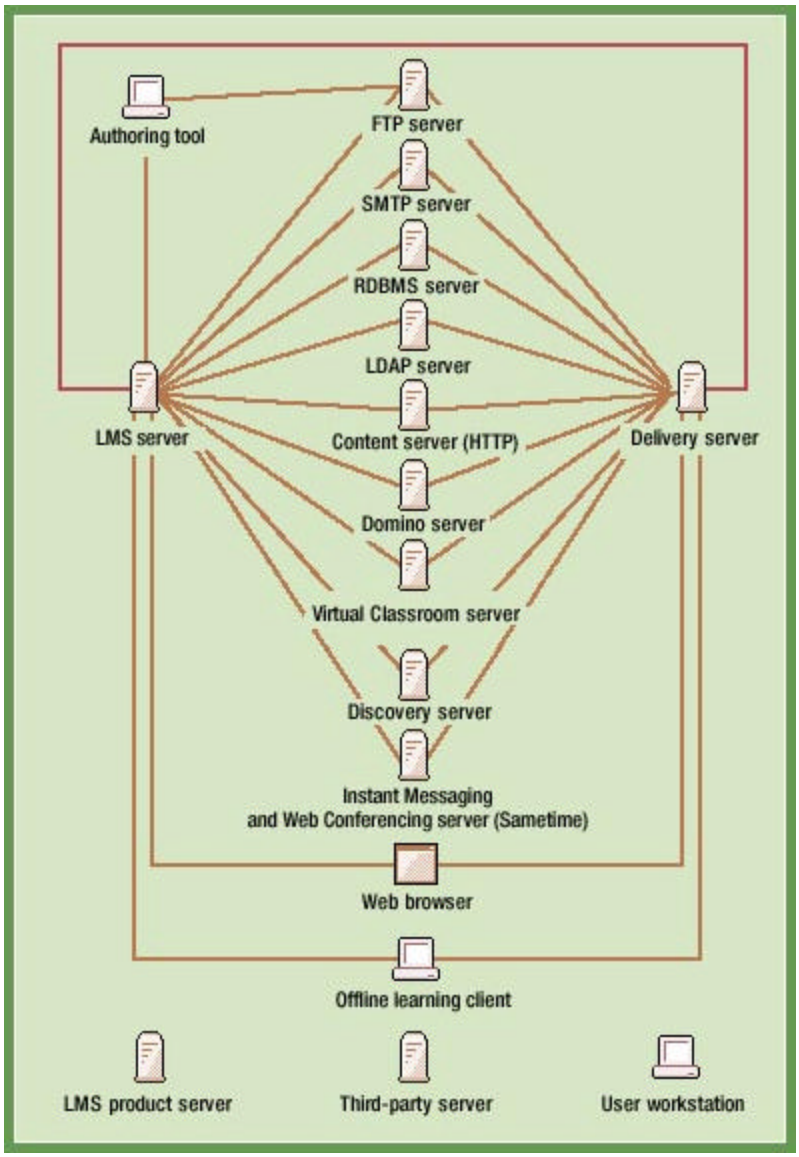
We considered a scenario complete when the following pieces were in place:

- A good objective and prerequisites
- Defined roles and tasks
- A test case doclink to support the tasks

We then subjected the scenario to peer review to ensure completeness. (We were not all experts in all functional areas.)

Scenario test hardware configuration

The following diagram shows a basic configuration for running our LMS scenarios. Bear in mind the configuration required can vary from one customer site to another, depending on available hardware:



Results

Our scenario test results were stored as Test Execution Results (TER) and Scenario Execution Results (SER). TERs tracked the results of all the test cases individually so any test "blockers" could be readily visible. SERs tracked the results of the entire scenario. For tracking purposes, TERs used the conventional Pass/Failed mechanisms. SERs used Passed, Failed, Failed with Errors, and Blocked. An SER marked as Failed meant that although there were some failed test cases, the tester could still complete the objective of the scenario. After issues were identified and fixed, the scenario was rescheduled into the test cycle. Failed with Errors meant that there were some failed test cases that were either Deferred or Not to be Fixed for this release to inform our management there were some issues with this activity. This confirmed that we did not need to reschedule the scenario for future testing.

After all scenarios had been through the review cycle and our team observed a consistent up-time and user load availability, we ran the scenarios against each supported platform. We divided our team into four subgroups with three testers per group. All four subgroups ran one scenario at a time, but all groups used the same server, giving us the advantage of getting more user load per machine with lots of simultaneous activity. This demonstrated the value add of scenario testing in addition to traditional feature testing for enterprise customers.

A very defined ship criteria for LMS 1.0 was that all scenarios had to achieve a status of Pass or Failed with Errors before we could ship. We adhered to this criteria and were successful at running all 20 scenarios. Of these, 19 scenarios Passed and one Failed with Errors. This gave us a good level of confidence that our users could perform a typical work day with LMS with few, if any, significant problems.

Conclusion

Based on our experience, we strongly encourage performing scenario testing after the first pass of functional testing (although you must first define a very solid set of Passed functional tests to avoid running into costly test blockers during the end cycle of the product). We found that successful scenario testing required hard work, dedication, and teamwork throughout the scenario system test cycle. But in the end, it was more than worth it because scenario testing allowed us to achieve focused results. Despite the occasional pain, we eventually got the hang of scenario testing, and (believe it or not) actually started having fun with it. We recommend you give scenario testing a trial run at your own site.

ABOUT THE AUTHOR

Linda Martineau is a QE Technical Lead in the LearningSpace Management System QE Group. She has been on the LMS product since early 2002 and is currently working on the Lotus Workplace Learning Center and the Personal Productivity Test Team. She is also a key technical person in the Ready For IBM LearningSpace program. Linda has been working in the QE field since the late 1980's.

