Commonly-Named

Sakai 11 LMS Testing

College of Charleston
CSCI362-01 Fall 2016
James Rhoten
Max Hyaska
Joe Lester

Table of Contents

Introduction

Sakai is a free, open-source Learning Management System (LMS). Performing much the same job as College of Charleston's OAKS application, Sakai allows for custom class lectures, assignments, resources, and schedules to be viewed by students and molded by teachers. The Sakai community is very active, and work together as teachers and developers to build upon the foundation of this education-focused software. The first thing we noticed about the Sakai 11 CLE software was its extensive documentation. Guides for installation, building, and testing the software were all available on the software's git repository. This made for a structured, while not necessarily easy, installation of the software.

Chapter 1: Installation

The installation of Sakai requires three programs to build and run. The provided guide walks through the installation of Java 1.8, Apache's Maven 3.2.5, and Apache's Tomcat 8 server. Once these programs have been installed with the correct environmental variable pointers, the source code was downloaded. We had to create a configuration file for Sakai and configure it to work with the prerequisite software. This was by far the hardest part. The guide provided relevant information to complete the process, but outside forums were referenced when complications arose.

Configuration

The software requires configuration of email and databases. This process was time consuming more than anything. Once the configuration was completed, building the software was relatively simple. The source code also provide sample data to build the software allowing for easy out of the box testing.

Testing

The documentation for Sakai 11 provides testing protocols and scripts. Testing information can be found here: https://confluence.sakaiproject.org/display/QA/Sakai+11+QA+Hub Here, Sakai provides tests and guides to making your own tests, as well as introducing a public weekly "Test Fest". Leaderboards display users who actively test and improve Sakai's software. Sakai uses Jira to document bugs and tests. We used a load testing framework provided by the Sakai documentation here: https://confluence.sakaiproject.org/display/QA/CLE+Load+Test+Framework. This testing framework runs through Maven, so a little research is required to configure and run the test suite.

Overall

Sakai was painful to download and build due to the unfamiliarity with Maven and Tomcat configuration. It is worth mentioning that a lot of the difficulty in setting up sakai seemed to stem from unfamiliarity with ubuntu. After the configuration and test data was set up, Sakai was ready to go. It was relatively easy to navigate, but complete functionality will take some time to observe. The power of the system will be difficult to tackle and maintain.

Chapter 2: The Testing Process

Sakai is large and has a lot of functionality for two types of users, teachers and students. We thought the most important aspects to test were those that applied to both users, the system requirements related to quizzes, including the creation by the teacher user, the participation by the student user, and grading performed by the system. Therefore, we will mainly be working with the test and quizzes tool of Sakai 11. We have created a database that holds sample data to allow us to access the roles of the entire simulated class.

Requirements traceability

One of the benefits of working with Sakai is the documentation. Our team figured this was the best place to look in order to tease out some of the requirements for this system. Below is the documentation we used to find some of these requirements.

Tests and Quizzes User Guide

Functional Requirements:

1. A teacher should be able to add, edit, and delete assessments through the Test and Quiz tool
2. The test and quizzes tool should allow for an instructor to view and update a quiz or test grade for individual students.
3. The test and quizzes tool should for the deletion of a quiz or test.
4. The test and quizzes tool should display a list of a quiz or test grades.

Nonfunctional Requirements:

1. The test and quizzes tool should allow the instructor to create quizzes and tests which include multiple types of questions.
2. The test and quizzes tool should sort the list of quiz or test grades according to student name, id, grade, submission time, and submission count.

Testing schedule

Sept 22 - Oct 2 -- test making \ Oct 3 - Oct 9 ---- test taking \ Oct 10 - Oct 16 -- test grading \ *Oct 18 ---------- deliverable due*

Hardware and Software Requirements

Sakai 11, Maven 3.2.5, Java 1.8, Tomcat 8, Ubuntu 14.04

System tests

---

test case ID: 01

test case description: A teacher user will create a 1 question, multiple choice quiz worth 10 points

related requirement(s): A teacher should be able to add, edit, and delete assessments through the Test and Quiz tool

pre-conditions: The user must be logged into an account with permission to change course content

test steps:

1. Navigate to the Test and Quizzes tool
2. Create an assessment with the title "MultChoiceQuiz"
3. Select multiple choice on "Insert New Question" drop down
4. Select question's point value as 10
5. Edit question text
6. Edit question answers
7. Confirm correct answer
8. Save assessment

expected results: A new, unpublished assessment named "MultChoiceQuiz" should be accessible through the Assessments tab

---

test case ID: 02

test case description: A teacher user will add a question to an unpublished, preexisting assessment

related requirement(s): A teacher should be able to add, edit, and delete assessments through the Test and Quiz tool

pre-conditions: The user must be logged into an account with permission to change course content. An unpublished assessment must be created and accessible.

dependencies: Test Case 01

test steps:

1. Navigate to the Test and Quizzes tool
2. Select unpublished assessment created in Test Case 1 named "MultChoiceQuiz"
3. Select True/False from "Insert New Question" drop down
4. Select question's point value as 10
5. Edit question text
6. Confirm correct answer
7. Save assessment

expected results: "MultChoiceQuiz" should now contain both a multiple choice and a T/F question.

---

test case ID: 03

test case description: A teacher will delete a pre existing assessment

related requirement(s): A teacher should be able to add, edit, and delete assessments through the Test and Quiz tool

pre-conditions: The user must be logged into an account with permission to change course content. An unpublished assessment must be created and accessible.

dependencies: Test Case 01

test steps:

1. Navigate to the Test and Quizzes tool
2. Find assessment named "MultChoiceQuiz" in Assessments tab
3. Select "Remove" from the "Select Action" dropdown
4. Confirm removal of assessment

expected results: "MultChoiceQuiz" should no longer be found under the Assessments tab

Chapter 3: Testing Framework

File Structure

/Commonly-Named

```
/TestAutomation

      /docs (contains all relevant documents, including README.txt, final report, presentation, and
poster
            README.txt

      /oracles (unused)

      /project (contains all necessary files from Sakai needed)
            /bin
            /src
                  StringUtil.java

      /reports (unused)

      /scripts (contains all scripts used by the project)
            runAllScripts.sh

      /temp (holds temporary files, such as the method outputs and the test report)
            Report.html
            Test01.txt
            Test02.txt
            ...

      /testCaseExecutables (contains all drivers used by the project)
            ContainsDriver.java
            ToStringCourseDriver.java
            ToStringStudentConstructorOne.java
```

```
            ToStringStudentConstructorTwoDriver.java
            ToStringUserDriver.java
            TrimLowerDriver.java

      /testCases (contains all test cases as .txt files)
            Test01.txt
            Test02.txt
            ...
            Test25.txt
```

The Driver

Each driver consists of one class file. Each class being tested has its own driver which will pass the parameters to the method being tested and save its output to a .txt file in the /temp directory. The hardest issue we faced was the vast labyrinth of dependencies that many of the classes within Sakai posses. We have expended a large amount of effort to ensure that the methods and classes being tested could be tested independent of the project itself. Many of the dependencies were from within the project itself, some are dependencies with networking and server connections, but few are only dependent on Java packages. These dependencies would require the installation of the entire Sakai program, Apache Tomcat, and Maven before the method could instantiated and tested.

We also ran into issues with our standardization of our test case files. The original template for our test cases did not provide an easy method of parsing and readability for others to examine or create new test cases.

Test Case Template

test case #

requirement being tested

CLASS.METHOD being tested

java driver file

expected outcome

inputs to the method (DELIMITED BY ",")

Chapter 4: Automation

Framework Changes

To better comply with the instructions provided for the project, we adapted our framework. A script was added to automate the testing suite, cycling through each test case to produce a final, comprehensive report of the tests. We implemented more test cases and identified multiple classes with methods to test, including User, Student, and StringUtil.

The Script

The script begins by deleting all files in the /temp directory. It then proceeds to compile all .java files in both the /project/src and /testCaseExecutables directories, ensuring that all relevant java files are executable. A template for a html table is formatted into a new file called Report.html. Next the script should read each .txt file in the /testCases directory line by line using a for loop. As an individual test case file is read, each line is placed into an element of a string array. The script should now gather the relevant information to run the driver, such as the driver's file name and the parameters to be passed. Once the driver has been executed, the script checks the /temp directory for a .txt with the same name as the test case. The file is read into the next empty element of the array and compared to the expected output defined by the test case. The result of this comparison is stored in the final element of the array. Finally, each array element is formatted into a html table row and added to Report.html located in the /temp directory. After the script has looped through each test case, it completes the Report.html and opens it in the default web browser.

Sample Test Cases

Test01

Search the list set to null for null element

StringUtil.contains

ContainsDriver.java

false

null:null

Test02

Test Empty least for element test.

StringUtil.contains

ContainsDriver.java

false

:test

Test03

Valid list containing one element which is being searched for.

StringUtil.contains

ContainsDriver.java

true

test:test

Test04

Valid list containing elements

StringUtil.contains

ContainsDriver.java

false

test1:test2

Test05

Valid list containing elements

StringUtil.contains

ContainsDriver.java

false

test7:test1,test2,test3,test4

Test06

Valid list containing elements

StringUtil.contains

ContainsDriver.java

true

Test3:test1,test2,test3,test4

Chapter 5: Fault Injection

Overview

The primary goal for this deliverable was fault injection. To do this we injected 5 different errors into the sakai source code that would cause test cases that would normally pass to fail. The primary purpose of this was to test that our script was in fact working and would report errors in the code should they be present, as opposed to just always stating that a test case passed. This also was useful for helping us better understand the code as it would not be possible to make the code fail in a specific way without a solid understanding of it. Another purpose of the fault injection would be to show that test cases were written correctly. Overall this was one of the easier deliverables of the project.

Important Concepts

The process of injecting the source code with faults is a pretty simple process, assuming one is familiar with how the code and drivers work. There are a few things that we found were important to consider. The first of these is that what ever fault one is trying to inject it should not cause the code to fail to compile. The test cases are not meant to test for syntax errors they are written to test functionality. If the code cannot compile then it not execute at all which won't help with testing. Syntax errors are best left to an IDE to keep track of. The next important aspect of this process is that one needs to have a firm grasp of how the code they are injecting errors into works. This will help in avoiding the previously mentioned concern.

Injection Process

The injection process started after we determined a few key points in the code that would cause the code to fail. We chose five places to place changes to the source code that would cause a failure. These were located in the StringUtil.java file, specifically to the contains and trimToLowerNull methods. Once these locations were chosen small changes were made to the code that would cause certain test cases to fail.

The Injected Faults

The first fault injected was located in the contains method, specifically around line 442.

Here we replaced:

```
if (value.length() == 0) return true;
```

with

```
if (value.length() == 0) return false;
```

This would cause an error when one searched an empty array for any string, which should return false as an empty string cannot contain an element by the very definition of what an empty string is. Because of this Test 2 failed as expected.

The Second fault was injected a few lines lower, at line 453

Here we replaced

```
if (value.equals((String) o)) return ;
```

with

```
if (value.equals((String) o)) return false;
```

Doing this would result in the method returning false when an array actually does contain the string it is being searched for. As expected tests 3 and 6 failed as they match the above criteria.

For the third fault injection we changed a return value around line 456.

Here we changed

```
return true;
```

to

```
   return false;
```

This change caused test 1, 4, and 5 to fail as they all reached the end of the collection which would mean the collection did not contain the desired item, but still returned true.

We placed the Fourth fault was placed in the trimToNullLower method at line 231:

Here we commented out

```
value = value.trim();
```

Which would cause the method to NOT remove any spaces before or after the string. This caused Test Cases 7, 9, and 11 to fail as they all have leading spaces.

The final Injection was near line 239.

At this spot we changed

```
return value.toLowerCase();
```

with

```
return value;
```

This would stop the method from turning uppercase letters into lowercase letter. This resulted in test cases 7, 9, 12, 13, and 14 to fail as they all had uppercase letters that needed to be changed to lower case.

Conclusion

As one can see the changes themselves were not too intensive, the results however, were quite dramatic. Out of our 25 test cases these injections caused 8 to fail. It was quite clear from these results that our testing automation framework is successful in its goals and that any test cases using these methods were constructed correctly, and more importantly tested correctly.

Chapter 6: Reflections

Time truly is the most valuable commodity.

Time management was a plague for our team. Between full-time jobs, internships, and classwork, it was very difficult to find shared free time to meet. As such, our team communicated mostly via e-mail and text. We spent an exorbitant amount of time trying to wade through the complexities of Sakai. We lost track of our schedules and plans, and soon found ourselves waist deep without a life raft. Once we dove into the project headfirst, things went smoothly, but our time table was cramped.

Github is a godsend

Github allowed our team to work on separate coding assignments with the ability to merge our code and test it out on our own schedules. Honestly, we neglected to realize and embrace the advantages offered by github early in the project, passing files around via e-mail. Once we were all aboard the Github train, our lives became much easier and less worrisome.

Teamwork

Our team rarely met, and deadlines were often neglected. Many long nights, pots of coffee, and gray hairs went into picking up the slack. We have grown to realize how the effects of procrastination by individuals affect the entire team. Working in groups is common in the software engineering field, thus students should strive to learn the most from their group projects, not only the software engineering aspect, but the human resource aspect as well. Lazy and unresponsive team members can decrease the moral and work ethic of the team, leading to sloppy and incomplete work.