System Requirements Specification

CSE 361 - Spring 2014

**Regression Testing System**

|  |  |
| --- | --- |
| Authors:  Alex Tobias  James Duin | Supervisor:  Dr. Greg Rothermel |

February 14, 2014

# **Table of Contents**

[Table of Contents](#_xjtwv3yydf1e)

[Revision History](#_sg4un0fqju49)

[1. Introduction](#_yng1l0oi8anu)

[1.1 Purpose](#_1whu1kractnu)

[1.2 Document Conventions](#_grz7s8d8vcv0)

[1.3 Intended Audience and Reading Suggestions](#_1ezz2msvlems)

[1.4 Product Scope](#_prlo5pqes0r6)

[1.5 References](#_r1thgguc72pf)

[2. Overall Description](#_v7q48a99jza5)

[2.1 Product Perspective](#_3ifu1d4gmsa)

[2.2 Product Functions](#_5ficg7itaoi)

[2.3 User Classes and Characteristics](#_aajha6h0pvm2)

[2.4 Operating Environment](#_si9u0yuguldh)

[2.5 Design and Implementation Constraints](#_a2gplbofwgrq)

[2.6 Assumptions and Dependencies](#_qmt6zij9pzx)

[3. Specific Requirements](#_7gs61a4vi24j)

[3.1 User Interfaces](#_siy11hy3shvf)

[3.2 Hardware Interfaces](#_rwd2p7fqzdf7)

[3.3 Software Interfaces](#_x6vfjgblhvo1)

[3.4 Communications Interfaces](#_i75apk7xv8jo)

[3.5 Phase I](#_x6p79izf2x5y)

[3.5.1 Description and Priority](#_9qtyt4wqdhoe)

[3.5.2 Stimulus/Response Sequences](#_1nvmcy66arxh)

[3.5.3 Functional Requirements](#_oe5mfckg3hxd)

[3.6 Phase II](#_l53pki48di)

[3.6.1 Description and Priority](#_87fdvl3f73v)

[3.6.2 Stimulus/Response Sequences](#_oyze2tmj25ke)

[3.6.3 Functional Requirements](#_iprd48lwln41)

[3.7 Future Release](#_o7eo7ebkc0ag)

[3.7.1 Description and Priority](#_itd4dk8ozqps)

[3.7.2 Functional Requirements](#_24ehfstuf97h)

[4. Other Nonfunctional Requirements](#_90ua7k8d5vid)

[4.1 Performance Requirements](#_ru9fcsmikmyv)

[4.2 Security Requirements](#_7xiqyze1kdws)

[4.3 Software Quality Attributes](#_h8cwfh53w3ly)

[4.4 Business Rules](#_bhbmsrvr2u95)

# 

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Alex Tobias & James Duin | 2-10-2014 | Initial draft. | 1 |
| Alex Tobias & James Duin | 2-14-2014 | Incorporate changes from peer review. | 1.1 |
| Alex Tobias & James Duin | 3-1-2014 | Updates to requirements for Design document. | 2 |

# **1.** **Introduction**

## 1.1 Purpose

The purpose of this document is to describe the requirement specifications for a Regression Testing System for software development teams. This document will cover the creation of the entire system and detail the initial two phases of the system along with features to be added in the future.

## 1.2 Document Conventions

The requirements will be grouped into the three phases of the product’s release, each requirement within a phase has its own priority in relation to other requirements in that phase. The list of requirements will show the priority of the requirements, that is, the higher the priority of the requirement the higher it is in the list. The definitions, acronyms, and abbreviations used in this document are as follows:

**QA** - Quality Assurance

**Test Case** - A browser automation script possibly written to be run by Selenium or Sahi.

**Test Suite** - A grouping of test cases, test case groupings are not mutually exclusive.

**Web app** - Web based application.

**IP address** - Internet Protocol address.

**Environment** - The url that the tests are to be executed against. This can be either point to a live site, or a development site that contains features are currently being worked on.

**CSE** - Computer Science and Engineering department at the University of Nebraska at Lincoln.

**VM** - Virtual Machine.

**Front end** - This references the web application portion of our software system.

**Back end** - This references the part of our software system responsible for executing the configuration completed by the user in the web app.

**XML** - Extensible Markup Language

**Jenkins** - Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle. Jenkins provides continuous integration services for software development.

**Sahi** - An open source browser automation tool.

**Selenium** - An open source browser automation tool.

**VirtualBox**- A virtualization software package for x86 and AMD64/Intel64-based computers from Oracle Corporation as part of its family of virtualization products.

## 1.3 Intended Audience and Reading Suggestions

This document is intended for the developers of this product, project managers, marketing staff, users, testers. For project managers, users, and marketing staff, we recommend reading the Product Scope, Product Functions, and User Interfaces sections and then the requirements breakdown in the System Features section. The Operating Environment, Software Interfaces, and Assumptions and Dependencies sections are meant primarily for testers and developers.

## 1.4 Product Scope

For our project, we will create a regression testing software platform for Quality Assurance (QA) testers. The user base would be QA testers that have groups of browser automation test cases but do not have a way of running those test cases on a schedule or on multiple browser/os combinations. The idea is the qa tester would access our regression testing web app and would upload the code for their browser automation test cases. They would then group the uploaded test cases into test suites via configuration pages on our website. On another configuration page they would specify the testing schedules they desire. These testing schedules consist of the following: the test suite, the time schedule they want the tests run, the browser/os combination, and the environment (live or dev url) they want to test on. The tests then will be run automatically for them and they will be able to see the test result logs on the website. They also will be able to see summary report pages that the website will generate from the test results. The reports will provide QA testers with the ability to track percent failure and test time for a testing suite over successive releases. It will not allow the user to run the tests on demand, or support more than one teams configuration. However, the system could be expanded to offer these capabilities in the future.

## 1.5 References

The following web addresses are references for the component open source software systems that will be integrated into our Regression Testing System:

1) Jenkins. (n.d.). *About Jenkins CI*. Retrieved Feb 9, 2014, from Jenkins: http://jenkins-ci.org/content/about-jenkins-ci

2) Sahi. (n.d.). *About Us | Sahi*. Retrieved Feb 9, 2014, from Sahi: http://sahi.co.in/about-us/

3) Selenium. (n.d.). *About Selenium*. Retrieved Feb 9, 2014, from Selenium: http://docs.seleniumhq.org/about/

4) VirtualBox. (n.d.). *Screenshots – Oracle VM VirtualBox*. Retrieved Feb 9, 2014, from VirtualBox: https://www.virtualbox.org/wiki/Screenshots

# **2.** **Overall Description**

## 2.1 Product Perspective

The Regression Testing System is a new self-contained system. It is not a component of a larger existing system. It is meant to satisfy the need of QA testers who may not have a VM stack setup with a lot of browser/os combinations. These testers need a way of setting up automatic testing schedules against a variety of such VMs. The Regression Testing System allows them to achieve this without committing any of their own computing resources to the task. All they have to do is upload the test case code into the web app, specify the test groupings and schedule, and check back on the results, the actual test execution and VM stack setup is abstracted away for them. The major components are as follows:

a) Web app - The User Interface for our system, run on a glassfish server, communicates with our Jenkins configuration

b) Jenkins - Continuous integration tool written in Java, communicates Selenium and VM

c) Selenium (or Sahi) - Browser automation software

d) VM - virtual machine, a software based emulation of a computer

## 2.2 Product Functions

* The system allows the user to upload browser automation scripts in Sahi or Selenium script code, they specify the name for the test case and it is saved for them within the web app.
* The system allows the user to group existing test scripts into test suites and save these test suite groupings.
* The system allows the user to save a testing schedule that consists of the following parameters: Test Schedule Name, Test Suite, Browser/OS, Environment, Weekly Schedule, Time of day.
* The system executes the test schedule and posts the results to a report page.
* The system allows the user to click on a test run in the report page and see the log files that were generated by Sahi or Selenium for that specific test run.

## 2.3 User Classes and Characteristics

The main class of user would be QA testers that would use the system execute browser automated regression testing for a web application on a schedule. The class of testers this product is aimed towards also does not want to set up their own VM stack. They also desire to be able to run the test suites on an automatic schedule. The system would also be ideal for testers who have a lot of existing test cases coded in Selenium or Sahi and desire a way to run the two scripts in the same testing suite. This class of testers would arise due to a qa department changing the software they use for browser automation, they would want to run both test case types without having to convert all of the test cases from the old system to the language used in the new system.

## 2.4 Operating Environment

The system involves a web app deployed to the CSE server, this will be public facing and can be accessed via the following url: <http://csce.unl.edu:8080/RegTesting>. The front end of the system will be contained within this web app, the back end can be setup on a separate pc. The pc must have jenkins, sahi, selenium, and VirtualBox installed and configured accordingly. The backend of the system is responsible for executing the tests on the specified schedule and returning the results to the web app, the interface between the frontend and backend.

The platform involves integrating existing open source tools for browser automation, virtualization and running test jobs, with a web application that we build ourselves. The platform contains the following components:

1) Jenkins: A continuous integration server. This will be used to kick off test cases on a schedule and deploy test results to our Regression web application. This will need to be installed on the testers pc and have access to the server the that hosts the web app. The QA tester will specify the schedule for the tests, the browser and machine that the tests will be executed on, and environment (live or dev url) that the tests will be run against. Jenkins will use shell scripts to execute the tests on the specified browser and machine. When the tests have concluded shell scripts in Jenkins will add any extra information to the test result xmls, such as machine and environment, and deploy the test results to the web app.

2) VirtualBox: An open source virtualization software package. This will be used to create several virtual machines that will be setup with different browser and OS combinations. This will also have to be installed on the testers pc and will be limited to Windows XP and linux instances as these are freely available.

3) Selenium and Sahi: Both are browser automation and scripting software. These will be used to code and run the test cases which will be started by Jenkins. These will need to be installed on the testers pc and Virtual machine instances. They will also generate the log files that will be modified and deployed by shell scripts within Jenkins. We will have to code some dummy test cases in the Sahi and/or Selenium script syntax just to demonstrate our application.

4) Regression testing web application: This will parse out information from the log files and store the information in a database, generate report pages. It should give QA engineers necessary metrics on regression testing over many releases of a product.

## 2.5 Design and Implementation Constraints

Since the backend operations of the web app will be on a different machine than the server that is hosting the web app, the two sides will have to communicate with XML data objects. The actual web page will be limited to using HTML, javascript, and java. The backend operations will be limited to using Jenkins. Also, since the delivered software is a public facing web app, we will have to maintain the delivered software for several browsers and operating system combinations.

The backend system involves setting up VM’s on a pc and that pc must have the VM’s running at the same time in order to satisfy the schedule the user specified. The number of vm browser/os combinations that the system will support is limited by the memory available on the pc.

The supported browser automation languages will initially be limited to two, Sahi and Selenium, but this could be expanded by future developers, as long as the browser automation software could be configured to get executed by jenkins on a desired VM.

The test schedules will have to be constrained to prevent errors would arise if test suites are configured in such a way that there is an overlap in the time it takes the test suite to run. For example if a test suite starts at 1:00 AM and takes 2 hours to run, and then the next test suite starts at 2:00 AM on the same machine then there will be an overlap. If these are the only two suites then there is not a problem since the second suite will just get pushed back to 3:00 AM, but the user could configure the system in such a way that the system never catches up and the backlog continues to build indefinitely. The system would no longer be executing the tests on the correct schedule. To resolve this there will have to be a requirement to warn the user, via text on the web app, if a backlog occurred

## 2.6 Assumptions and Dependencies

The assumed factor for this project is mainly the project’s dependency on Jenkins, Selenium, and Sahi to be reliable tools for our page to use. We assume any automatically generated code inside the IDE we will use (Eclipse, NetBeans) will be correct and work properly. Our system is dependent on integrating Jenkins, Sahi, Selenium, and VirtualBox softwares with a java web app. The project would be affected if any of these softwares changes and no longer supports the expected features. Jenkins must support shell scripting, Sahi and Selenium must support executing tests via shell and ant targets. VirtualBox must support running VMs at the same time and a unique network IP address for each VM.

# **3.** **Specific Requirements**

## 3.1 User Interfaces

The following images show the front end of the system, these pages will be accessible from the web app and any interaction the user has with the system will take place through the web app.

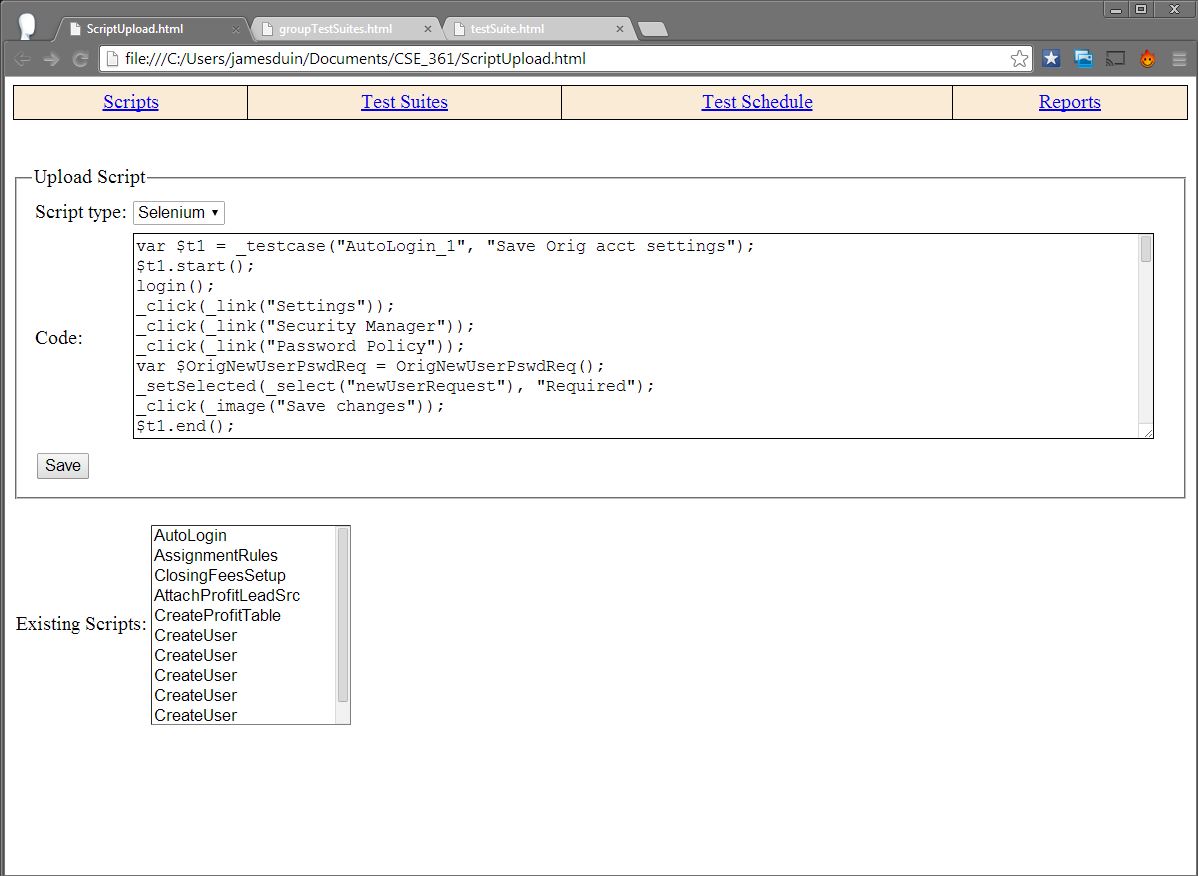


Figure 1: Page to upload a script.

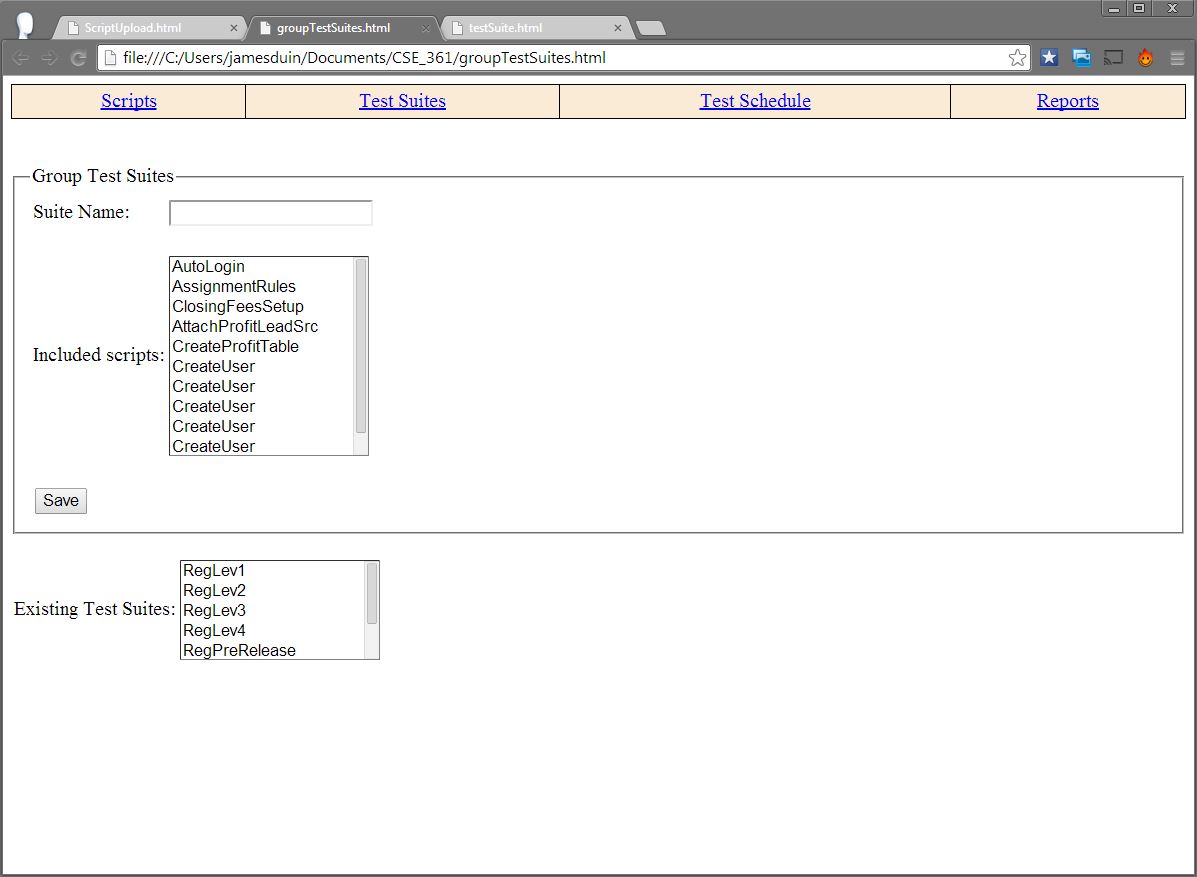


Figure 2: Page to group scripts into test suites.

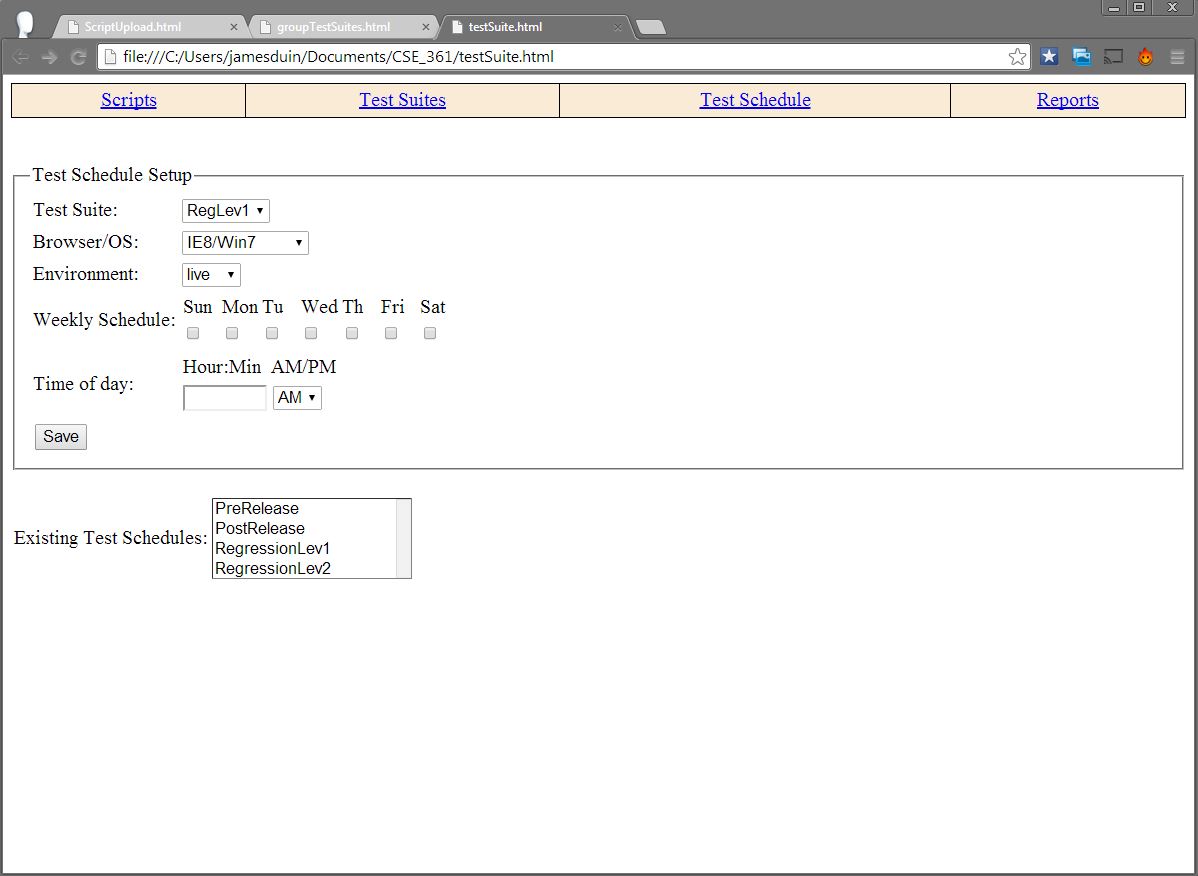


Figure 3: Page to specify a testing schedule.

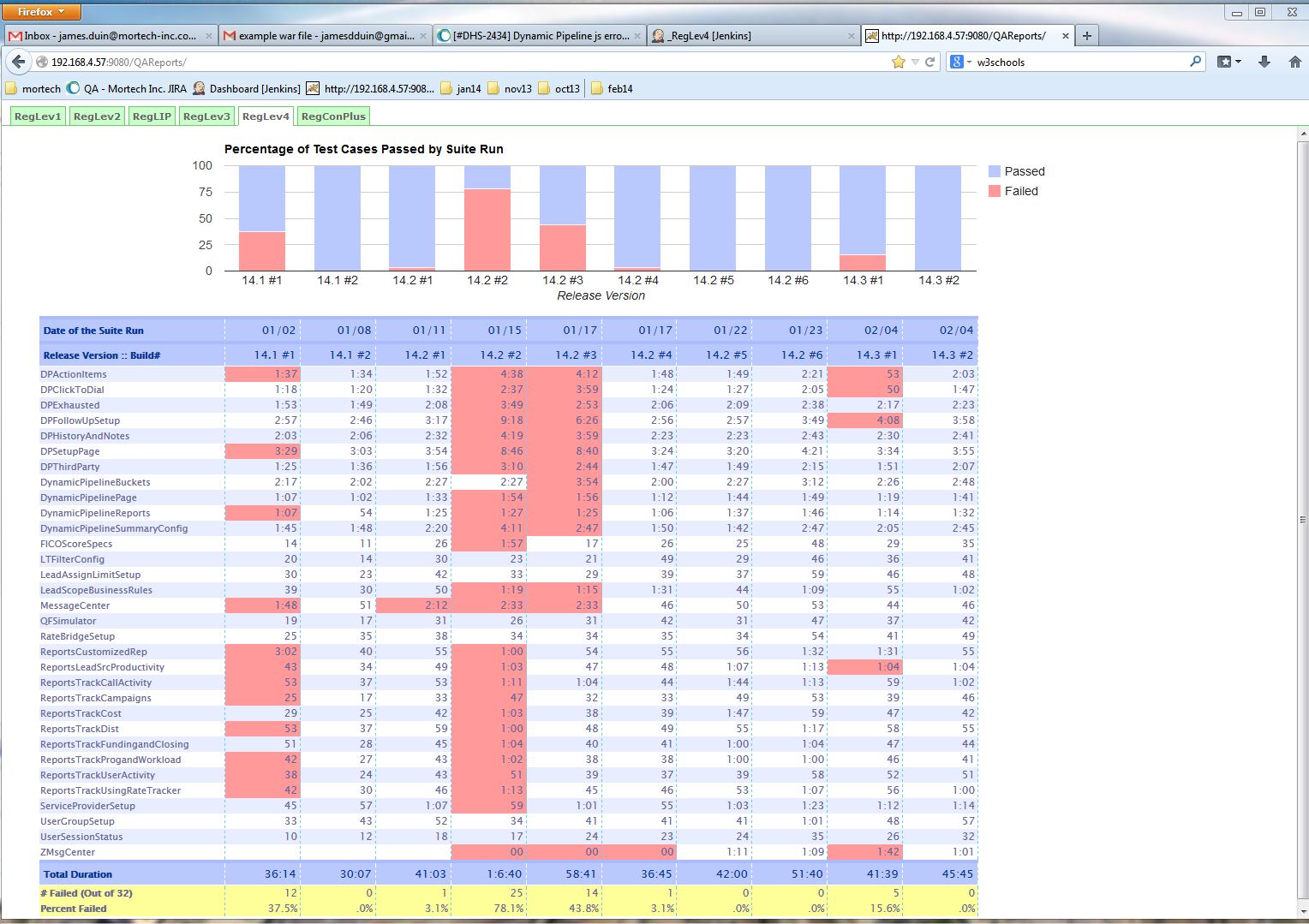


Figure 4: Report pages that show the results of regression testing from successive releases.

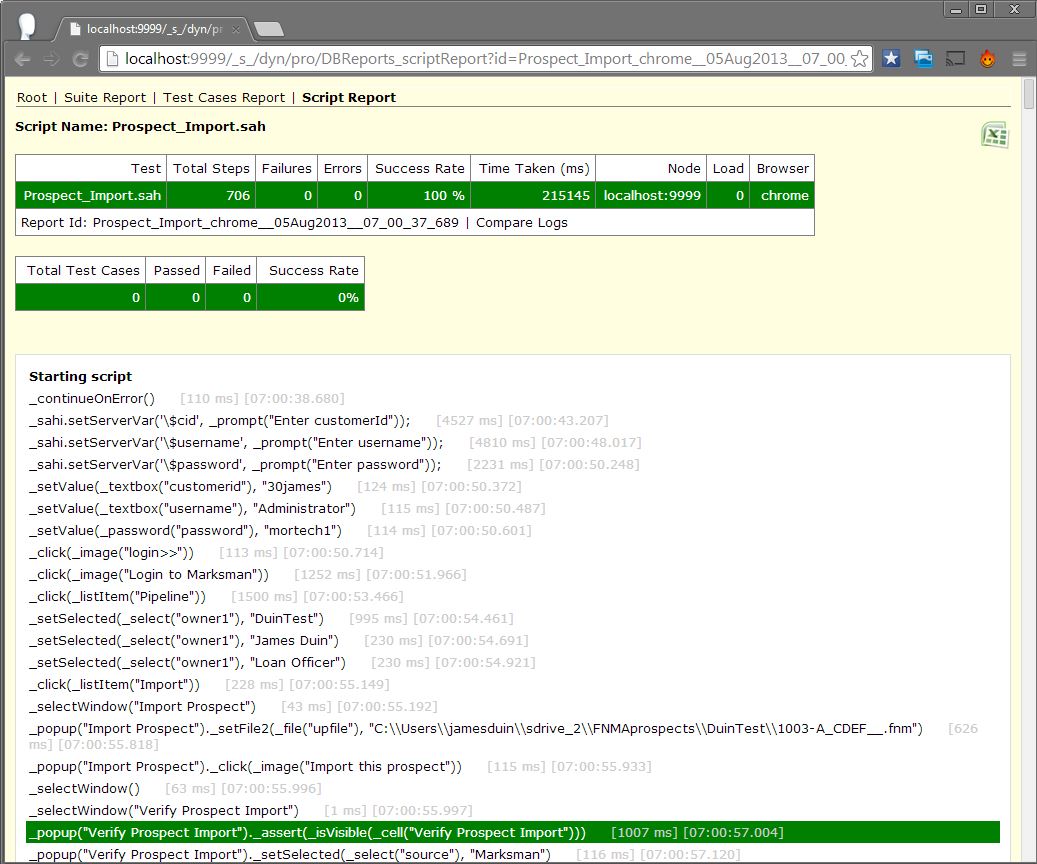


Figure 5: Sahi generated html log for an individual test case showing the step by step execution of the test.

## 3.2 Hardware Interfaces

Our system will work on a pc that has a functional web browser installed and a reliable internet connection. Supporting a mobile version of the website is not required.

## 3.3 Software Interfaces

The main software interface is between the web app front end and the back end system on a pc. Jenkins tasks will be responsible for integrating the web app with the browser automation softwares and virtual machines. We will use Jenkins to execute shell scripts on the test schedule specified on the web app and to post the test results back to the web app. This will require setting up ‘jobs’ to execute the following tasks: download any new test case code and save code into the necessary directory on the backend, create the actual test case ‘jobs’, create the actual test suite ‘job’, update the test suite ‘job’ with a schedule and start the test suite runs on that schedule, check the website for new test cases, new suite groupings and updates to the test schedules and kick off the appropriate ‘job’ to execute the configuration update. It may take time for jenkins to update the backend systems so we will restrict the test schedules to over 30mins into the future, that way we can guarantee the backend systems will be updated by the time the test suite is meant to be executed.

## 3.4 Communications Interfaces

The web app must be visible to the machine that is running the back end. Any environment that a user specifies in the web app configuration must also be visible to machine running the backend. To send data from the web page to the Jenkins configuration, our web app will use XML objects.

## 3.5 Phase I

### 3.5.1 Description and Priority

For the first phase of our system we will provide only the highest priority features, so as to demonstrate the most important aspects of the product functioning. The main features are the ability for the user to upload code for a test case in Sahi and have that script be executed on a virtual machine on the specified schedule and the test results be accessible to the user on the reports page.

### 3.5.2 Stimulus/Response Sequences

The user will upload the code for a test case in Sahi, and enter in all of the test configurations on the web app. When the test case is ready, the user will click ‘Save’ on the web page. This will trigger an event which grabs all of the data given by the user, formats it into an XML object, and posts it for Jenkins to see. Jenkins will be set up to check for new test case objects every 5 minutes, and will grab the XML object, and use the information to configure the test case. Then Jenkins will run the test case and produce a new XML object with the test case results, which it will then post back to the web page. Then, on the reports page of the web app, it will parse through the provided XML object and fill out the page with the test case data.

### 3.5.3 Functional Requirements

REQ-1.1: The system shall allow a user to upload a single script in Sahi code via a Scripts page, this script will be saved and should be accessible whenever navigating to the Scripts page.

REQ-1.2: The system shall allow a user to save a test schedule for the single Sahi script on IE9/Win7.

REQ-1.3: The system shall execute the test on the desired test schedule and post the test results to the report page.

REQ-1.4: The system shall allow the user to click on the a test result from the reports page and access the Sahi generated step by step html log for that test.

## 3.6 Phase II

### 3.6.1 Description and Priority

The second phase will involve expanding the product to allow for multiple scripts and the ability to group those scripts into test suites. It will also involve expanding the product to support Selenium test cases. Multiple VMs will need to be added so as to support the following browser/os combinations: IE8,Chrome,Firefox on XP; IE9,10,11, Chrome, Firefox on Win7; IE10,11, Chrome, Firefox on Win8.1.

### 3.6.2 Stimulus/Response Sequences

The user will upload the code for a test case in Sahi, and enter in all of the test configurations on the web app. When the test case is ready, the user will click ‘Save’ on the web page. This will trigger an event which grabs all of the data given by the user, formats it into an XML object, and posts it for Jenkins to see. Jenkins will be set up to check for new test case objects every 5 minutes, and will grab the XML object, and use the information to configure the test case. Then Jenkins will run the test case and produce a new XML object with the test case results, which it will then post back to the web page. Then, on the reports page of the web app, it will parse through the provided XML object and fill out the page with the test case data.

### 3.6.3 Functional Requirements

REQ-2.1: The system shall allow a user to upload and save multiple scripts in Sahi code on the Scripts page.

REQ-2.2: The system shall allow a user to access the saved script code by selecting an existing script from on the scripts page and loading the code for that script onto the page.

REQ-2.3: The system shall allow a user to group the existing saved scripts into test suites, saving the test suites on a test suite page.

REQ-2.4: The system shall allow a user to save multiple test schedule for a test suite on one of the many supported browser/os combinations. These combinations including: IE8,Chrome,Firefox on XP; IE9,10,11, Chrome, Firefox on Win7; IE10,11, Chrome, Firefox on Win8.1.

REQ-2.5: The system shall prevent the user from saving a test schedule within 30 minutes (in the future) of the current time.

REQ-2.6: The system shall shall execute the test suite on the desired browser/os combination and post the result back to the web app.

REQ-2.7: The system shall allow the user to access a report page showing the last 10 runs for a given test suite.

REQ-2.8: The system shall detect when an overlap in test runs has occurred and display a warning detailing the suites in conflict on the web app.

## 3.7 Future Release

### 3.7.1 Description and Priority

These are possible ways to expand the system to offer additional support to QA testers that are not to be implemented in either the first or second phase.

### 3.7.2 Functional Requirements

REQ-3.1: The system shall execute a test schedule on Safari/OS X.

REQ-3.2: The system will allow multiple QA teams to login to the web app and only access configurations saved on that account.

REQ-3.3: The system shall allow a user to upload and save multiple scripts in Selenium code on the Scripts page.

# **4.** **Other Nonfunctional Requirements**

## 4.1 Performance Requirements

The user does not need to be able to run a test immediately, in fact the only way they can run tests through our system is by grouping them into a test suite and specifying a schedule for that test suite to run.

## 4.2 Security Requirements

The phase I & II requirements will result in a publicly accessible website without any login credentials so there are no security requirements at this time. Anybody can access the configuration and save/upload code to be run. Future releases will have to expand the product to serve accounts in order for this security risk to be resolved.

## 4.3 Software Quality Attributes

The test results must be verifiable as the application is meant to be used by testers. The accessibility of step by step log reports from the web app allows the QA tester to see exactly what caused the failure.

## 4.4 Business Rules

Anyone who has access to this web app can utilize the full functionality of the software system.