**San Jose State University**

**Computer Engineering Department**



CMPE 287 - Software Quality Assurance

**REGRESSION TEST PLAN**

**Project – Elevator System**

**Instructor:**

**Dr. Jerry Zeyu Gao**

**Submission Date:**

21 September, 2011

**Submitted By**

Sonalika Gupta (sonalika.gupta@gmail.com) SJSU ID: 007490509

Anshu Basia (anshu.basia@gmail.com) SJSU ID: 006447987

Venkata Nagasri Lavanya Davuluri SJSU ID: 008091083

(lavanya\_davuluru@yahoo.com)

Shubhada Narkar (shubhada\_narkar@yahoo.com) SJSU ID: 008137337

Table of Contents

[List of Figures 4](#_Toc304168172)

[List of Tables 5](#_Toc304168173)

[1. Introduction 6](#_Toc304168174)

[1.1 Project Overview 6](#_Toc304168175)

[1.2 Project Goal 7](#_Toc304168176)

[1.3 Document Structure and Scope 8](#_Toc304168177)

[1.4 Software Test cycle 9](#_Toc304168178)

[1.5 Evaluation Criteria 10](#_Toc304168179)

[2. Requirement Analysis 10](#_Toc304168180)

[2.1 Component Description 10](#_Toc304168181)

[2.2 Functional Requirements 11](#_Toc304168182)

[2.3 Non-Functional Requirement 11](#_Toc304168183)

[2.4 Use Case 12](#_Toc304168184)

[3. Targeted Testing and Criteria 13](#_Toc304168185)

[3.1 Testing Strategy 14](#_Toc304168186)

[3.2 Testing Process 14](#_Toc304168187)

[3.3 Testing Approach 15](#_Toc304168188)

[3.3.1 Component/Unit Testing 15](#_Toc304168189)

[3.3.2 Graphical User Interface Testing 16](#_Toc304168190)

[3.3.3 Integration Testing 17](#_Toc304168191)

[3.3.4 System Level Re-testing 18](#_Toc304168192)

[3.3.5 Functional Testing 19](#_Toc304168193)

[3.3.6 Configuration Testing 19](#_Toc304168194)

[3.3.7 Performance Testing 20](#_Toc304168195)

[3.3.8 Automation Testing 20](#_Toc304168196)

[3.4 Testing process after adding New Components - Alarm Indicator and Alarm Switch 21](#_Toc304168197)

[3.4.1 Component and Integration Testing 21](#_Toc304168198)

[3.4.2 System Level Re-testing 21](#_Toc304168199)

[3.4.3 System Level Regression and Configuration Testing 23](#_Toc304168200)

[3.5 Testing Criteria 23](#_Toc304168201)

[3.5.1 Black Box Phase 23](#_Toc304168202)

[3.5.2 White Box Phase 23](#_Toc304168203)

[3.5.3 Regression Phase 23](#_Toc304168204)

[4. Re-Testing Requirements 24](#_Toc304168205)

[4.1 Hardware Requirements 24](#_Toc304168206)

[4.2 Software Requirements 24](#_Toc304168207)

[5. Re-testing Tasks and Schedule 24](#_Toc304168208)

[5.1 Roles and Responsibilities 24](#_Toc304168209)

[5.2 Project Deliverables and Schedule 27](#_Toc304168210)

[5.3 Testing Schedule 27](#_Toc304168211)

[5.4 Individual Testing Plan 28](#_Toc304168212)

[6.1 Elementool 32](#_Toc304168213)

[6.2 JUnit 35](#_Toc304168214)

[7. Test Standards 39](#_Toc304168215)

[7.1 Standards 39](#_Toc304168216)

[7.2 Defect Reporting 39](#_Toc304168217)

[7.3 Defect Tracking Process 40](#_Toc304168218)

[7.3.1 Defect discovery/defect naming 40](#_Toc304168219)

[7.3.2 Defect resolution 41](#_Toc304168220)

[7.3.3 Process improvement 41](#_Toc304168221)

[8. Assumptions/Constraints 42](#_Toc304168222)

[9. Summary 42](#_Toc304168223)

[10. References 43](#_Toc304168224)

[11. Team Member’s Contribution 43](#_Toc304168225)

[Appendix A 44](#_Toc304168226)

[1. Test case Template: 44](#_Toc304168227)

[2. Test case result Template 45](#_Toc304168228)

[3. Defect Reporting Format 46](#_Toc304168229)

# List of Figures

Figure 1: Existing Component Structure Diagram – Version 2 7

Figure 2: Component Diagram for the enhanced Elevator System – Version 3 8

Figure 3: Use case diagram 13

Figure 4: Testing process after adding new components to the system 22

Figure 5 : Gantt chart for Test schedule 28

Figure 6: Test case screenshot from ELEMENTOOL 32

Figure 7: Test report screenshot from ELEMENTOOL 33

Figure 8: Bug reporting screenshot from ELEMENTOOL 34

Figure 9: Issue report screenshot from ELEMENTOOL 34

Figure 10: JUnit test case screenshot 35

Figure 11: JUnit Test Suite execution screenshot 35

Figure 12 : Selenium Firefox add-on screenshot 36

Figure 13 : Selenium test case recording screenshot 37

Figure 14 : Selenium Test Suite execution screenshot 38

Figure 15: Bug Life Cycle 40

# 

# List of Tables

Table 1: GUI Testing Information 16

Table 2: Integration Testing Information 18

Table 3: System Level Re-Testing Information 19

Table 4: Functional Testing Information 19

Table 5: Configuration Testing Information 20

Table 6: Performance Testing Information 20

Table 7: Automation Testing Information 21

Table 8: Roles of Team members 24

Table 9 : Responsibilities of role assigned 26

Table 10: Project Deliverables and Schedule 27

Table 11: Testing Schedule 28

Table 12: Individual Testing Plan 31

Table 13: Defect Details in defect log 41

# 1. Introduction

## 1.1 Project Overview

Regression testing is software testing carried out to detect any new errors that might have crept in the existing functionality after changes have been made to it. These changes may be enhancements, configuration changes etc. Regression testing of software involves re-running previously executed test cases and verifying that program behavior has not changed in any unwarranted manner. This can be achieved systematically and efficiently if we select the appropriate set of test cases which would cover the change adequately.

This Regression Test Plan Document is for “Component Based Elevator Simulation System” which has been developed in Java and follows a component-based model.

This test plan will also cover Configuration Testing for the Elevator system. The configuration testing will test the system with each of the configurations of software and hardware that are supported.

This elevator simulation system comprises of following modules currently:

* Admin Console
* Algorithm model that drives the entire system
* Meta controller
* Car Component
* Car Controller
* User panel
* User panel queue
* Door and its operational Components
* Door Panel
* Floor panel
* Floor Panel Queue

An elaborate component diagram for the modules of Elevator System mentioned above is shown below.

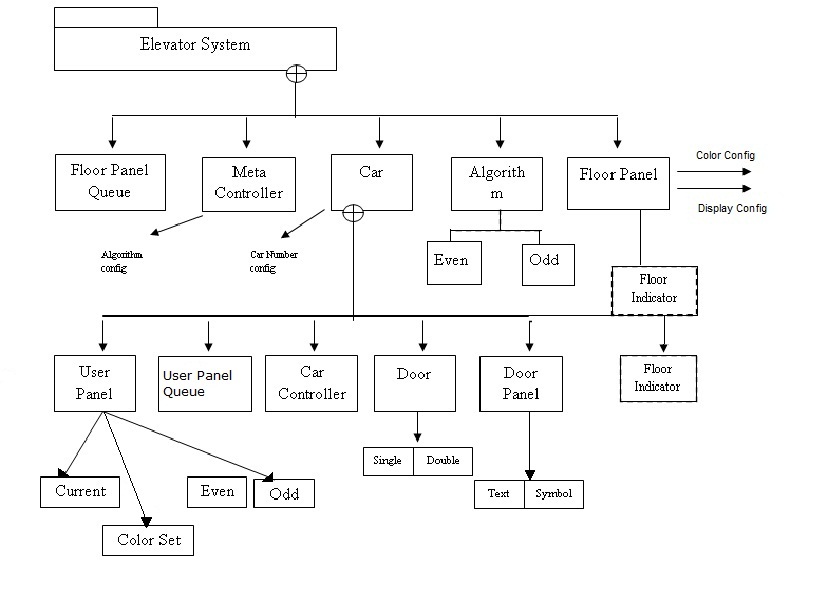
****

Figure 1: Existing Component Structure Diagram – Version 2

## 1.2 Project Goal

Following enhancements will be done to the existing elevator system:

1. A new component internal Alarm component will be added by the developers of our team. Floor panel component will be modified to accommodate this component.
2. An external Alarm component with an indication function on each floor for an elevator system will be added.
3. Another algorithm or elevator protocol will be added for elevator car.
4. Also, the development team will enhance the existing configuration user interface to support the selection and configuration of the newly added Alarm function and features.

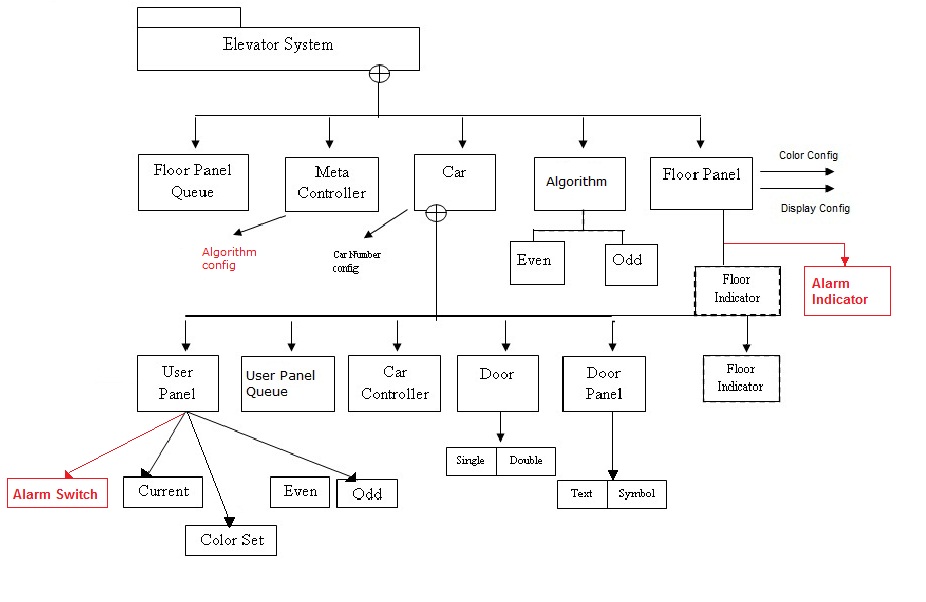


Figure 2: Component Diagram for the enhanced Elevator System – Version 3

(Changes have been marked in red)

## 1.3 Document Structure and Scope

This Regression Test Plan document provides a systematic approach for regression testing of the Elevator System after the enhancements to discover bugs or errors after the suggested changes have been incorporated. This document defines the scope of testing along with the methodologies and strategies that will be used in testing the system. It also specifies the roles and responsibilities of the team members along with the resources and testing tools that shall be used during the entire testing phase along with the schedules that shall be followed.

Following components will be under the scope of testing in this test plan:

New Components:

* Alarm Indicator Component
* Alarm Switch Component
* Algorithm to support alarm function and movement of elevator car

Affected Components:

* User Panel
* Floor Panel
* Car
* Algorithm
* Meta Controller

Component –level testing for these components will be carried out using:

1. Black Box testing:

* Equivalence Partition.
* Decision Table Testing

1. White Box testing:

* Basis Path Testing

System-level regression testing will use following testing methodology:

1. Black box testing method

* Equivalence partition
* Decision table

## 1.4 Software Test cycle

Following steps will be followed in the Software Test Cycle in the entire duration of the project:

1. Test Requirements

* Requirement Specification documents
* Trace-ability Matrix for ensuring Test Coverage

2. Test Planning

* Test Scope, Test Environment
* Different Test phase and Test Methodologies
* Manual and Regression Automation Testing
* Defect tracking tools

3. Test Environment Setup

4. Test Design

* Traceability Matrix and Test coverage
* Test Scenarios Identification & Test Case preparation
* Test case reviews and Approval

5. Test Execution and Defect Tracking

* Executing Test cases
* Testing Test Scripts
* Capture, review and analyze Test Results
* Raise the defects and tracking for its closure

6. Test Reports and Acceptance

## 1.5 Evaluation Criteria

Test cases should be designed meticulously to ensure no test requirement is missed. This shall be achieved by developing a Requirement Traceability Matrix which will map the test cases to the requirement they correspond to.

Addition of Alarm component will result in modifications to other components which shall be regress tested by reusing some of the existing test scenarios along with the new test cases designed to cover the changes done.

All the test cases selected from the existing test-bed and the newly designed test cases should be executed successfully with expected result same as the execution outcome. The defects shall be logged in using a bug tracking tool along with the details of scenario and details of steps to reproduce it.

# 2. Requirement Analysis

## 2.1 Component Description

* Alarm Switch

In the User Panel of elevator Car, a new feature Alarm Switch needs to be added. This Alarm Switch component will be a configurable feature with two options. In case of emergencies, user will be able to press the Alarm Switch and call for attention. The components affected by this change will be User Panel, Car and Meta Controller.

* Alarm Indicator

This will be an external component with an indication system on each floor for the elevator system. Alarm Indicator will indicate the floor where the lift is stuck in case of emergencies when the Alarm Switch is pressed. Floor Panel component will be affected by the introduction of Alarm Indicator component.

* Algorithm for elevator system

The elevator system is already equipped with two algorithms – Shortest Path and Random Path algorithms.

A new algorithm will be added to the current set of algorithms for the elevator system to control the movement of the Elevator Car. Elevator Controller will need to be updated to support this newly added algorithm.

## 2.2 Functional Requirements

* Add an internal Alarm component with a switch function in the user panel of each elevator Car (for your selected elevator system). Please make sure to provide two different models for your Alarm component to allow your customers to select.
* Add an external Alarm component with an indication function on each floor for an elevator system. You can decide what kind of indication function.
* Update your floor panel component to accommodate the external Alarm component.
* Add one more elevator protocol (or algorithm) to control the movement of your Elevator Car, and update your Elevator Controller to support this as one of options.
* Enhance the existing configuration user interface to support the selection and configuration this Alarm function and features.

## 2.3 Non-Functional Requirement

Non-functional requirements of a system are criteria used as a quality check of the system after the enhancements or code changes have been done. The non-functional requirements or the quality criteria for the system after enhancements and regression testing are:

* Accuracy- Accuracy or correctness is a very desirable non-functional requirement. It is expected that the elevator system will provide as accurate or correct a service as before the changes.
* Maintainability- Maintainability is the ease with which maintenance of a system can be carried out according to the prescribed standards. This is to ensure that the changes made to the elevator system do not affect the maintainability of the system in any way.
* Reliability- Reliability is measured as the probability of failure or in other words, ability of a system to perform for a specified duration under the prescribed conditions. It can be measured as Mean Time Between Failure (MTBF). The target here is to maintain the reliability of the elevator system after the code changes have been incorporated.
* Response time- Response time for the elevator system is from the time a request has been made to the time it got in action to serve the request. For example- if a user presses Door Close button, then acceptable response time would be 3 seconds i.e. the door should start closing in 2 seconds.
* Testability- Software testability of a system is the level of testing it supports in a given context. If a system is not very testable then the test effort involved would be of higher degree. Testability of a software system is highly dependent on the requirements, size and complexity of the software.

## 2.4 Use Case



Figure 3: Use case diagram

# 3. Targeted Testing and Criteria

The main components of the elevator system are as follows

1. Admin Console
2. Algorithm model that drives the entire system
3. Meta controller
4. Car Component
5. Car Controller
6. User panel
7. User panel queue
8. Door and its operational Components
9. Door Panel
10. Floor panel
11. Floor Panel Queue

New components to be added to the system

* Alarm Indicator
* Alarm Switch
* Algorithm to support alarm function and movement of elevator car

Affected components

* User Panel
* Floor Panel
* Car
* Algorithm
* Meta Controller

## 3.1 Testing Strategy

1) ***Black Box testing***:

1. *Equivalence Partition*

Equivalence partition testing involves the technique of partitioning the entire input domain into different disjoint partitions such that all the elements in one partition have all or none property. This technique reduces the testing effort by minimizing the need to execute redundant test cases, which produces the same output. But, this testing does not ensure complete testing coverage and hence other types of testing should also be used to ensure the quality of product.

**Coverage:** Tests the functionality of the component.

1. *Decision Table Testing*

Equivalence partition testing is useful when the variables under analysis are mutually independent. To test scenarios, where combinations of action are taken and they depend on various logical relationships, decision table testing is the best choice. Hence, instead of choosing other black-box testing methods, we opted for decision table method.

**Coverage:** Tests the functionality of the component in complex scenarios by combining various conditions and actions based on them.

**2) *White Box testing***

* 1. *Basis Path Testing*

A program or component of a system can be represented as a control flow graph. Nodes of the graph represent program statements and directed edges represent flow control. This testing technique focuses on testing all plausible paths of the flow graph while removing the redundant test cases.

**Coverage:** The basis path testing executes every statement of program at least once while providing all edge and node coverage.

## 3.2 Testing Process

* Analyze the system and its requirements.
* Identify the components to be tested.
* Select the test methods to be used.
* Apply the test methods and develop test cases.
* Execute test cases developed in previous step.
* Record the result of test cases.
* Compare the result of test cases with the expected output.
* Report the bugs using bug tracking tool.

## 3.3 Testing Approach

This section defines the approach that will be adopted for testing and re-testing the system and will help in attaining high quality of the system

### 3.3.1 Component/Unit Testing

The smallest compilable entity of the system is called unit or component. It is basically an atomic unit and cannot be decomposed further. Testing of such small entity is called component testing. This type of testing is done at component level to ensure their proper working. Unit testing will be done for all the new component as well as for all those components, which will be affected by the new addition. Unit testing of Alarm Switch and Alarm Indicator as well as of Car Controller and User panel will not only help in rectifying bugs at an early stage but also reduces complications at the later stage of integration and performance testing.

Testing Tool to be used for unit testing is **JUnit.**

Methods to be used for component testing are:

1) White Box testing – Basis Path testing

2) Black Box Testing – Equivalence Partition Testing and Decision Table Testing

This type of testing is basically conducted by developer to make sure all the components are working correctly and produce expected output and also confirm to the requirements of the system. In this project, besides doing unit testing on new components, we will retest all those components affected by new components to ensure nothing breaks or behave differently after the addition.

### 3.3.2 Graphical User Interface Testing

As the name suggests, this testing technique is used to test graphical user interface. GUI is very important part of any system or product and its proper functioning and representation is vital for any system. GUI testing ensures the following

1) The graphical layout of the system is in accordance to the laid requirements.

2) All the GUI controls are working properly and are placed in correct position.

3) Events generated by different GUI components call proper functions.

4) Results of any GUI interaction is displayed correctly and properly and in the manner desired as specified in the requirement and design document.

5) Proper display of error messages.

Thus, GUI testing basically verifies the interaction between user and the system interface. This is done through a variety of test cases and the test suite generated for such testing should cover all the functionality of the system and should make sure if the test suite fully exercises the GUI or not.

In our system, we will be adding Alarm Indicator and Alarm Switch component to our system. This will call for some changes to User Panel, which acts as an interface for user interaction. Hence, we decided to go for GUI testing.

|  |  |
| --- | --- |
| Test Objective: | To make sure that the change to graphical user interface does not affect the existing functionality and new changes work properly according to requirement specification. |
| Technique: | Manual Testing: Manually creating a test suite.  Automation Testing :  Tool : Selenium (Selenium is an automated GUI Test Framework) |
| Coverage: | Functional ( domain and interface flow) |

Table 1: GUI Testing Information

### 3.3.3 Integration Testing

Integration testing basically consists of two steps integration and testing. In this phase of software testing, individual software components are grouped together and testing is performed on them collectively. This testing is performed after unit testing but before validation testing. Integration test plan is made and then software modules, which have already been individually tested using unit testing are grouped together in large aggregates and then test cases as defined in the plan are applied to them. The output of this phase of testing is integrated system, which is ready for system testing. The purpose of integration testing is to check performance, function and reliability requirements mentioned in design specifications. Generally, Black Box methods are used to test the system integration in which output from one component is fed as input to another component and different components are winded together. This type of testing is usually carried out by test engineers and developers.

|  |  |
| --- | --- |
| Test Objective: | To find out inconsistency between modules, in their interfaces and in functional requirements of already tested components |
| Technique: | Black Box methods:   * Equivalence Partition Method * Decision Table Method |
| Coverage: | Components that will be aggregated together for integration testing are as follows:   1. Alarm Indicator, Car Controller 2. Alarm Switch, User Panel 3. Alarm Switch, Alarm Indicator 4. Alarm Indicator, Alarm Algorithm 5. Alarm Algorithm, Algorithm 6. Alarm Algorithm, Meta Controller 7. Alarm Indicator, Floor Panel 8. Elevator Movement Algorithm, Algorithm 9. Elevator Movement Algorithm, Meta Controller |

Table 2: Integration Testing Information

### 3.3.4 System Level Re-testing

System testing is performed on entire system and is done in the context of System Requirement Specification (SRS) or Functional Requirement Specification (FRS). Once integration testing is complete, the complete, integrated system moves on to next phase of testing i.e. system testing. This testing checks the system on the whole for requirements specified at the requirement stage and desired output. This testing requires no knowledge of inner code or design of the system. It is done by test engineers.

|  |  |
| --- | --- |
| Test Objective: | The focus is to find defects within the integrated/aggregated components and also within the system as a whole. |
| Technique: | Black Box Testing methods such as state based testing. |
| Coverage: | To test functionality, usability, performance, security, input/output, behavior, recovery, user acceptability of the system as a whole |

Table 3: System Level Re-Testing Information

### 3.3.5 Functional Testing

This type of testing emphasizes on testing function and behavior of the component. Functional testing verifies the component against design or requirement specification while system testing validates the system against system requirement. This testing is performed by test engineers.

|  |  |
| --- | --- |
| Test Objective: | The focus is to verify the component against design and requirement specifications |
| Technique: | Black Box Testing methods will be used. |
| Coverage: | * All newly added components will be tested for functionality i.e. Alarm Switch, Alarm Indicator. * Impacted components like Car, User Panel, Floor Panel, Algorithm will be functionally re-tested. * GUI of User Panel as well as of Admin Console will be re-tested to ensure proper functioning. |

Table 4: Functional Testing Information

### 3.3.6 Configuration Testing

Configuration testing intends to test software operability in different environments and configurations. Developer and tester can never know, the system they are building will be run by client in what type of environment and on what hardware and software combinations. Hence, the aim of this testing type is to test the system for its robustness to run on various combinations and under different environment.

|  |  |
| --- | --- |
| Test Objective: | To check the system with each possible configuration of hardware and software. This test will also test the various customization options and the working of the system based on options selected. |
| Technique: | To be decided. |
| Coverage: | Test the whole system on different platforms and combinations of hardware and software and various customization options available for the system. |

Table 5: Configuration Testing Information

### 3.3.7 Performance Testing

Performance of a software system is a very broad aspect and can involve two types of testing technique qualitative testing and quantitative testing. Quantitative testing measures attributes like measuring the response time and Qualitative testing measures attributes like reliability, scalability, interoperability, etc.

|  |  |
| --- | --- |
| Test Objective: | To check the performance of the system under different conditions. |
| Technique: | To be decided. |
| Coverage: | Whole system will be tested for reliability. |

Table 6: Performance Testing Information

### 3.3.8 Automation Testing

Testing without manual intervention is called automation testing. Here, all the test cases are executed with the help of software or tools and then expected outcomes are compared with the expected result.

|  |  |
| --- | --- |
| Test Objective: | To execute all test cases using some tools or scripts, without manual intervention. |
| Technique: | To be decided |
| Coverage: | To generate automated test suite for whole system |

Table 7: Automation Testing Information

## 3.4 Testing process after adding New Components - Alarm Indicator and Alarm Switch

### 3.4.1 Component and Integration Testing

Alarm Indicator and Alarm Component are new components to be added to the system. Once their development is complete, developer itself will do the unit testing using black box and white box methodologies to test their proper functioning and behavior. If any bug is discovered by the developer, appropriate changes are made to the code in order to fix it. Also, changes are made to the existing components in order to incorporate the new components to the system. These changes are also once tested by the developer at the unit level to ensure proper working.

Once unit testing on developer side is complete, the new components and impacted components will be passed to the test engineer for further testing. The test engineer will determine the components affected by the new components and will first prepare test cases for unit testing. The test cases will be executed for every component passed to the tester. If any bug is found, a bug report will be generated and concerned developer or developing team will be notified. Once, the unit testing is complete, the next step is integration testing. Again, a new set of test cases will be prepared by the test engineer and components will be tested in group. If any bug is found, a bug report will be generated and concerned developer or developing team will be notified. Otherwise, system level re-testing process will be initiated. If any bug is found, the developer will fix the bug and pass on the fixed components again to the tester and whole process will be repeated again.

### 3.4.2 System Level Re-testing

Once the system has been integrated, the test engineer will move on to system level testing. For this, the test engineer can re-use already existing test cases, create new test cases and can also discard any unused old test-cases. The test engineer will test the system as a whole for proper functionality as specified in the requirement specification and will also check if any old part of the system is not broken due to new changes. If any bug is discovered, it is recorded and passed onto the developer for further changes in the system.



Figure 4: Testing process after adding new components to the system

### 3.4.3 System Level Regression and Configuration Testing

Once, system level re-testing is complete, regression and configuration testing starts which test the system for complete functionality and various qualities like customization, security, reliability, throughput, performance, etc.

## 3.5 Testing Criteria

### 3.5.1 Black Box Phase

*a) Entry Criteria:* It depends on user requirement and component specification. Once the requirements are clear and development is complete, black box testing at unit level can be initiated.

*b) Exit Criteria:* Successful execution of all the planned test cases and comparison of results with expected one will lead to exit point. Thus 100% success rate is required to exit black box phase.

### 3.5.2 White Box Phase

*a) Entry Criteria:* White box testing is performed on individual components. Hence, its entry criteria depend on program structure, program internal logic, data structures, program internal behavior and states of a program.

*b) Exit criteria:* A success rate of 90% is required to exit white box testing phase.

### 3.5.3 Regression Phase

*a) Entry Criteria:* Regression testing starts after integration testing. Thus, all the components should be integrated properly into one whole system for regression testing to begin.

*b) Exit Criteria:* Once all the test cases have been executed and test results have been compared and approved, one can exit regression testing phase.

# 4. Re-Testing Requirements

## 4.1 Hardware Requirements

Intel Pentium based PCs

## 4.2 Software Requirements

Operating Systems: windows XP/VISTA/7

Browser: IE, Mozilla Firefox

Word Processor Software-MS Office 2007

Eclipse IDE for JAVA

JUnit and GUI Testing Framework

Elementool

# 5. Re-testing Tasks and Schedule

## 5.1 Roles and Responsibilities

**Roles**

|  |  |
| --- | --- |
| **Role** | **Team Member** |
| Test Manager | Anshu Basia |
| Test Engineers | Sonalika Gupta, Venkata Davuluri, Shubhada Narkar |
| Test Automation Engineers | Sonalika Gupta |
| Developers | Anshu Basia, Venkata Davuluri |

Table 8: Roles of Team members

**Responsibilities**

|  |  |
| --- | --- |
| **Role** | **Responsibilities** |
| **Test Manager** | * To supervise and coordinate all the project management related activities. * Identify and define the regression test project scope. * Deciding upon test strategies and testing tools for the project. * Assigning the responsibilities to test engineers. * Reviews deliverables prepared by the test team. * Initiating team meetings and coordinating between testers and developers. |
| **Test Engineers** | * Analyzing and understanding the project requirements and the GUI of the system. * Preparing the test cases and test plan. * Peer- review of test cases. * Preparing Black-box test cases and executing them. * Identifying regression test cases and executing them. * Performing manual testing. * Documenting test results, bug reports, and test project summary in a manual test report. |
| **Test Automation Engineer** | * Setting up the automation framework for the testing project. * Automating the regression test cases by using appropriate Record and Playback tool identified for the project. * Executing the automated Regression test cases. * Preparing automation bug report. |
| **Developers** | * Understanding the requirements and designing of the components based on the specified requirements. * Implementing code changes for the components and coding new components as per the requirements in the existing application. * Performing unit testing by following the methodology specified for white-box testing tool. * Interacting with the test engineers in case of doubts and concerns. |

Table 9 : Responsibilities of role assigned

## 5.2 Project Deliverables and Schedule

|  |  |  |
| --- | --- | --- |
|  | **Milestone** | **Date** |
| 1. | Regression Test Plan Document | 09/21/2011 |
| 2. | Regression Test Design Specification | 10/12/2011 |
| 3. | Manual Regression Test Document | 10/26/2011 |
| 4. | Automatic Regression Testing Report | 12/07/2011 |
| 5. | Configuration Test and Complexity Analysis Report | 11/30/2011 |
| 6. | Project Demo | 12/07/2011 |

Table 10: Project Deliverables and Schedule

## 5.3 Testing Schedule

To ensure that all the tasks are delivered on time, we tried to plan ahead and drafted a schedule that details the start and end dates and the approximate time to be spent on each task.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Start Date** | **End Date** | **Duration** |
| Regression Test Plan Document (Initial draft) | 09/05/11 | 09/12/11 | 7 days |
| Regression Test Plan Document (Final draft) | 09/13/11 | 09/20/11 | 7 days |
| Regression Test Design Specification (Initial draft) | 09/25/11 | 10/03/11 | 8 days |
| Regression Test Design Specification (Final draft) | 10/04/11 | 10/11/11 | 7 days |
| Manual Regression Test Document (Initial draft) | 10/13/11 | 10/21/11 | 8 days |
| Manual Regression Test Document (Final draft) | 10/21/11 | 10/25/11 | 4 days |
| Configuration Test and Complexity Analysis Report (Initial draft) | 11/01/11 | 11/18/11 | 18 days |
| Configuration Test and Complexity Analysis Report (Final draft) | 11/19/11 | 11/29/11 | 10 days |
| Automatic Regression Testing Report (Initial draft) | 11/30/11 | 12/03/11 | 3 days |
| Automatic Regression Testing Report(Final draft) | 12/04/11 | 12/06/11 | 2 days |
| Project Demo | 12/01/11 | 12/06/11 | 11 days |

Table 11: Testing Schedule



Figure 5 : Gantt chart for Test schedule

## 5.4 Individual Testing Plan

|  |  |  |
| --- | --- | --- |
| **Team Member** | **Task** | **Time** |
| Anshu Basia | * Deploy the source code and understand the Elevator system. * Add new component to the existing elevator system as per the change requirement * Update components that are affected by the new requirement * Run the Black box test cases for Alarm component * Test case design – White box methods for Alarm component * Run the White box test cases for Alarm component * Perform System testing for Elevator system * Perform Performance testing for Elevator system * Perform Automatic regression testing for the Elevator system | 09/10/11 – 09/17/11  09/18/11 – 09/23/11  09/24/11 – 09/27/11  09/28/11 – 10/01/11  10/02/11 – 10/08/11  10/09/11 – 10/14/11  10/15/11 – 10/25/11  11/01/11 – 11/26/11  11/28/11 – 12/05/11 |
| Venkata Davuluri | * Deploy the source code and understand the Elevator system * Add new component to the existing elevator system as per the change requirement * Update components that are affected by the new requirement * Run Black box test cases for Alarm component * Test case design – White Box methods for the Floor panel component * Run the White box test cases for the Floor panel component * Perform Performance testing for the elevator system * Perform Function testing for the elevator system * Perform Automatic regression testing for the Elevator system | 09/10/11 – 09/17/11  09/18/11 – 09/23/11  09/24/11 – 09/27/11  09/28/11 – 10/01/11  10/02/11 – 10/08/11  10/09/11 – 10/14/11  10/15/11 – 10/25/11  11/01/11 – 11/26/11  11/28/11 – 12/05/11 |
| Sonalika Gupta | * Deploy the source code and understand the Elevator system * Test case design – Black Box methods for the new Alarm component * Run the Black box test cases for the Alarm component * Test case design – White Box methods for the Alarm component * Run the White box test cases for the Alarm component * Perform System Testing for the Elevator system * Perform Configuration Testing for the Elevator system * Perform Automatic regression testing for the Elevator system | 09/10/11 – 09/17/11  09/18/11 – 09/25/11  09/26/11 – 10/01/11  10/02/11 – 10/08/11  10/09/11 – 10/14/11  10/15/11 – 10/25/11  11/01/11 – 11/26/11  11/28/11 – 12/05/11 |
| Shubhada Narkar | * Deploy the source code and understand the Elevator system * Test case design – Black Box methods for the Floor panel component * Run the Black box test cases for the Floor panel component * Test case design – White Box methods for the Floor panel component * Run the White Box test cases for the Floor Panel component * Perform System Testing for the Elevator system * Perform Configuration testing for the Elevator system * Perform Automatic regression testing for the Elevator system | 09/10/11- 09/17/11  09/18/11 – 09/25/11  09/26/11 – 10/01/11  10/02/11 – 10/08/11  10/09/11 – 10/14/11  10/15/11 – 10/25/11  11/01/11 – 11/26/11  11/28/11 – 12/05/11 |

Table 12: Individual Testing Plan

6. Testing Tools

## 6.1 Elementool

Elementool ([www.elementool.com](http://www.elementool.com)) provides various web based tools for issue tracking, time tracking, help desk, etc.

We can write test cases and assign them using the test case tool.

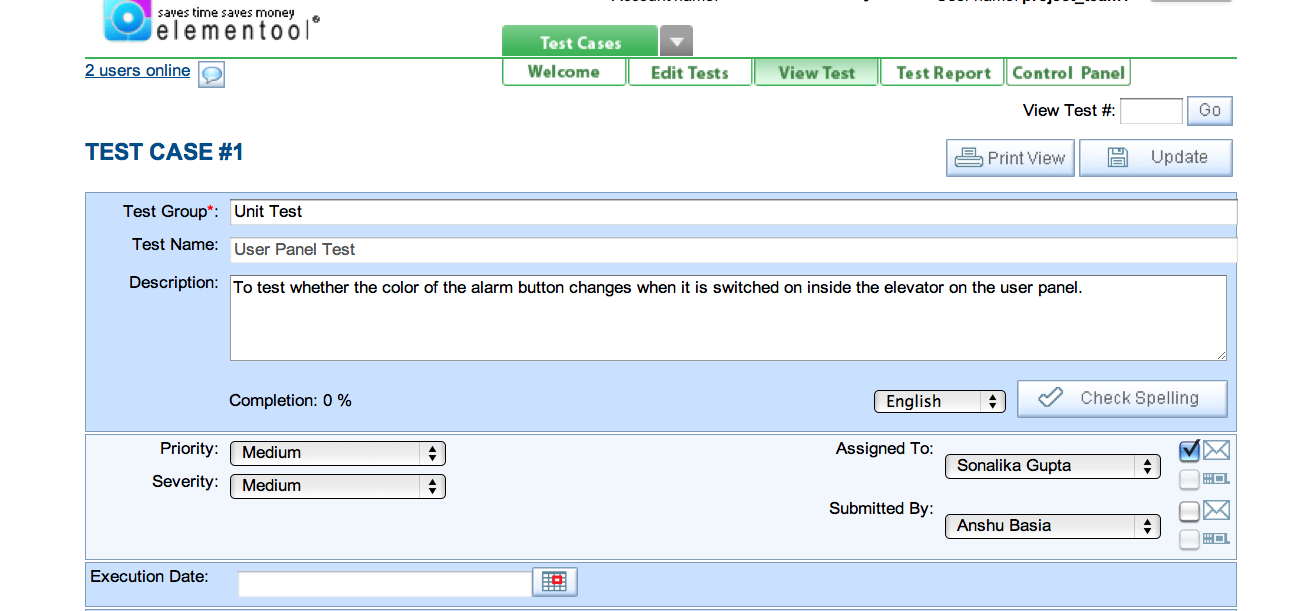


Figure 6: Test case screenshot from ELEMENTOOL

A test report can be generated to view all the test cases along with their priorities and to whom they are assigned.

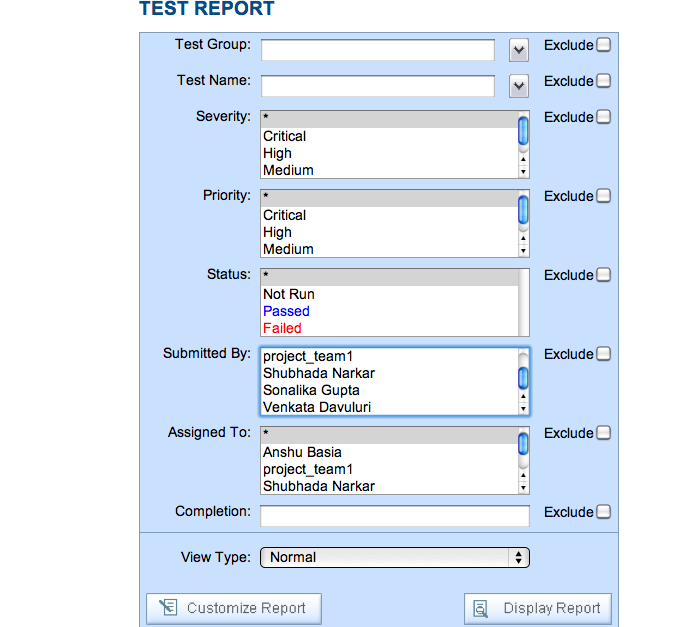


Figure 7: Test report screenshot from ELEMENTOOL

Bugs can be filed online in the issue tracking tool and an email will be sent to notify the person to whom the bug was assigned.



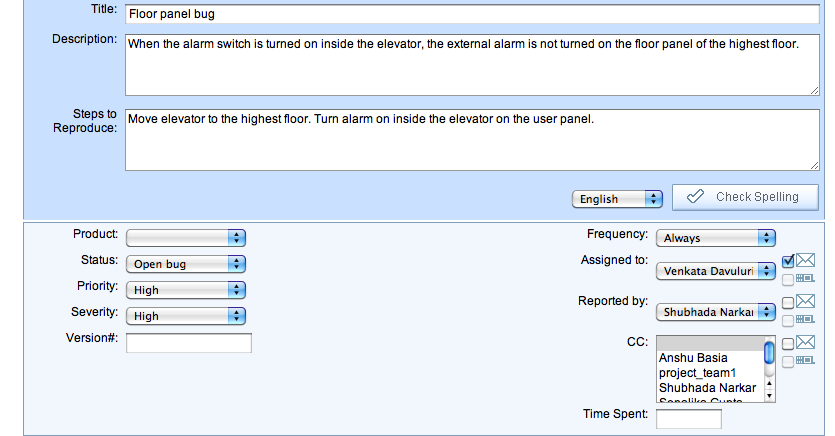


Figure 8: Bug reporting screenshot from ELEMENTOOL

The Project manager can track all the issues by generating the issue report with all the desired criteria. A history of all the bugs is also maintained.

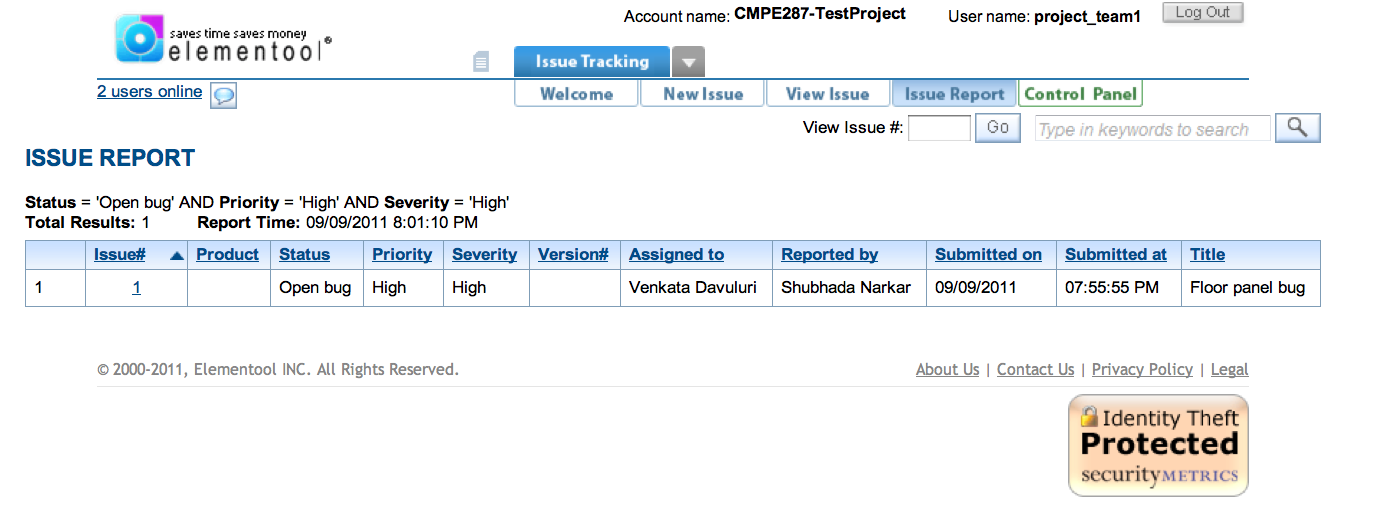


Figure 9: Issue report screenshot from ELEMENTOOL

Elementool can be used also for time tracking, file sharing, etc. Everything is online and only an account needs to be created to access all the tools. One account for multiple users will suffice i.e. all the team members can be added to a single account.

## 6.2 JUnit

JUnit (www.junit.org) is an open source framework useful for running unit test cases (in Java). It is free of cost. It can be used by developers to write test cases and test their code while developing. Tests are not inter-dependent. Many tests can be run at a time independently.

We can write test cases for our classes and run them.

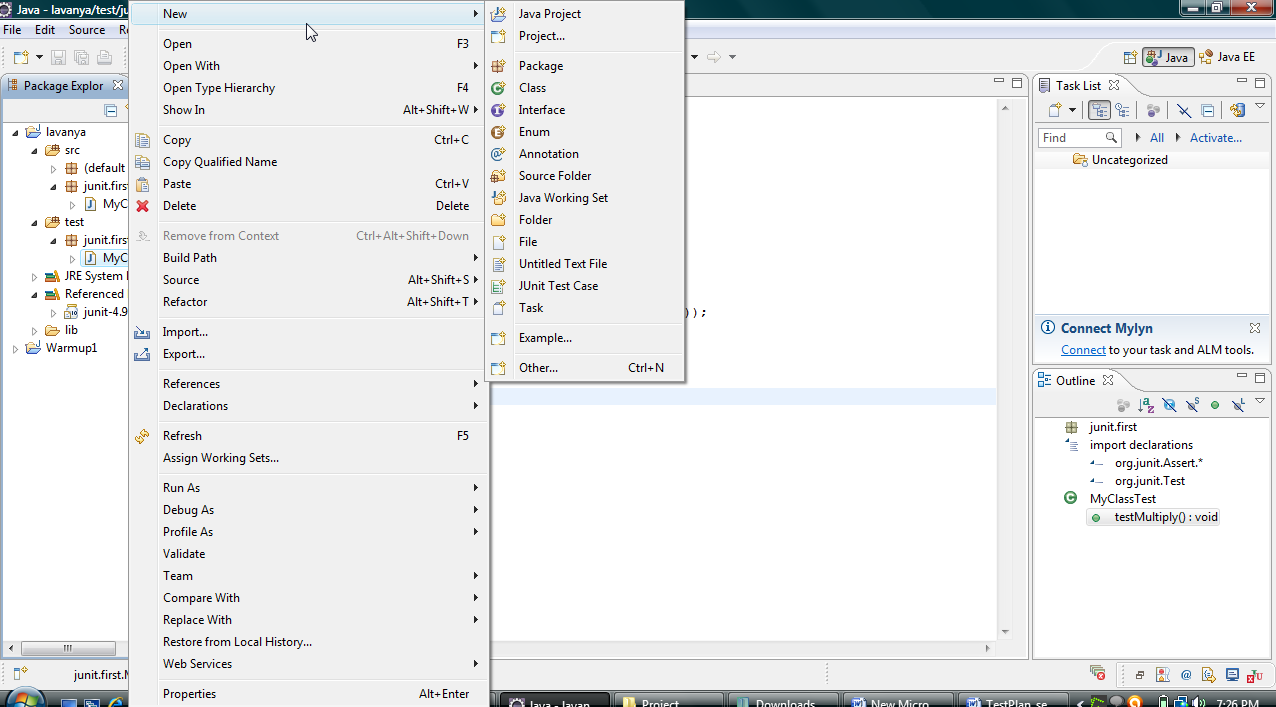


Figure 10: JUnit test case screenshot

Once the test case is executed on the target source code, a report of success or failure of the test is produced. We can also have many test cases in a Test suite and run them.

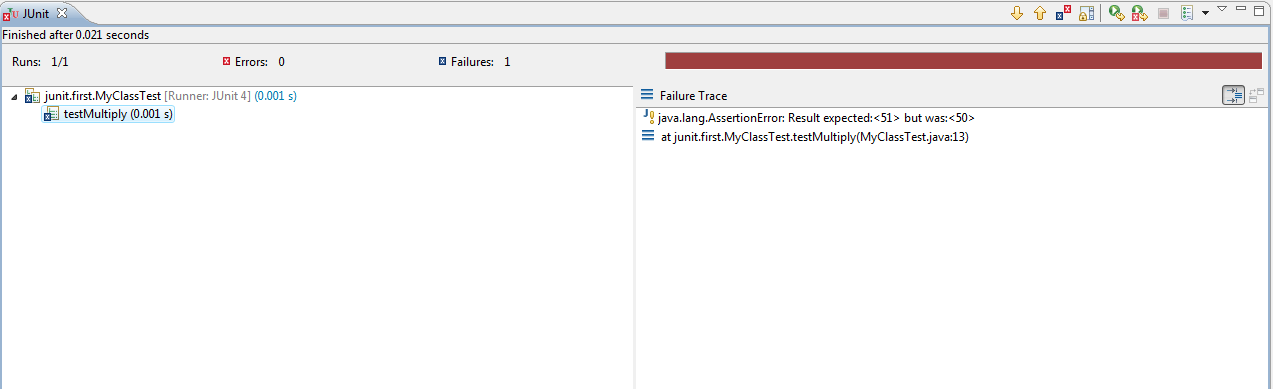


Figure 11: JUnit Test Suite execution screenshot

**6.3 Selenium**

Selenium (<http://seleniumhq.org>) is a functional testing tool to test GUI components. It can be downloaded as an add-on for the Firefox browser.

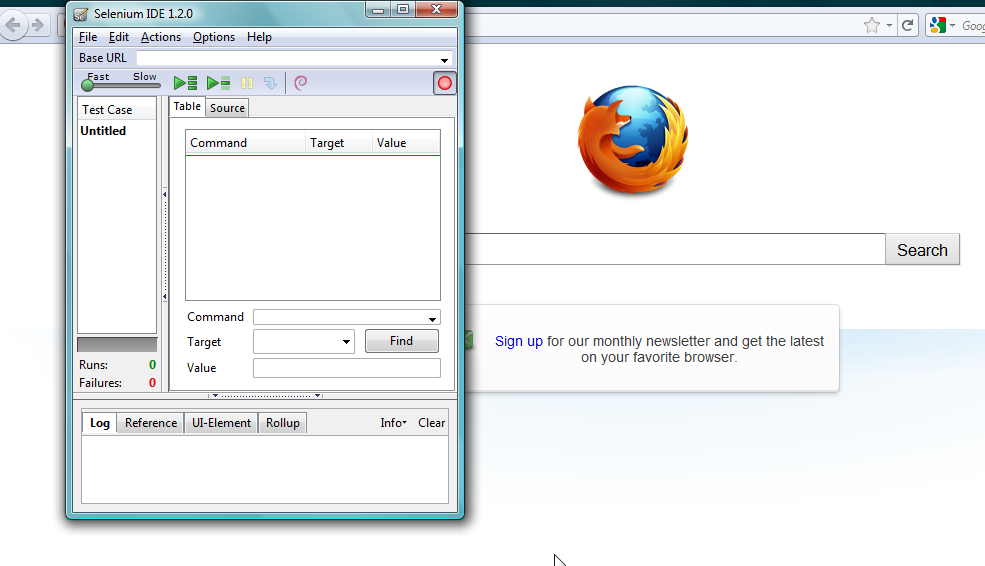


Figure 12 : Selenium Firefox add-on screenshot

Simple test scripts can be written to run a test case or we can record the actions and play them back to run the test cases.

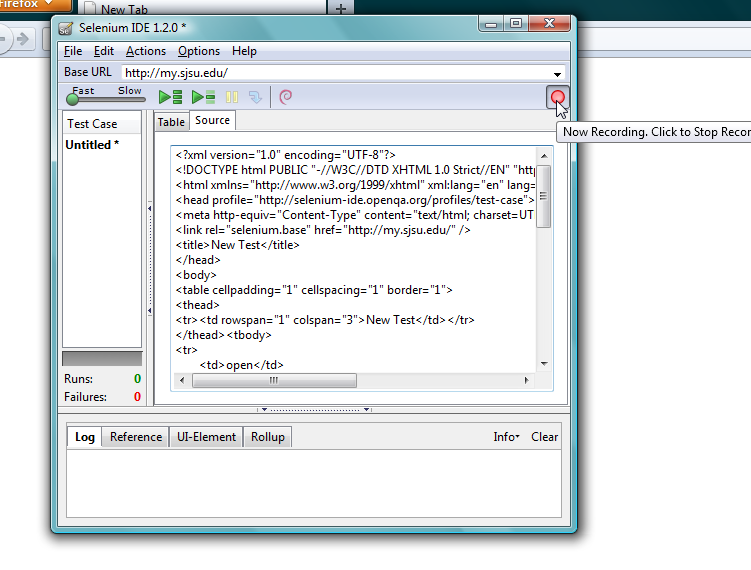


Figure 13 : Selenium test case recording screenshot

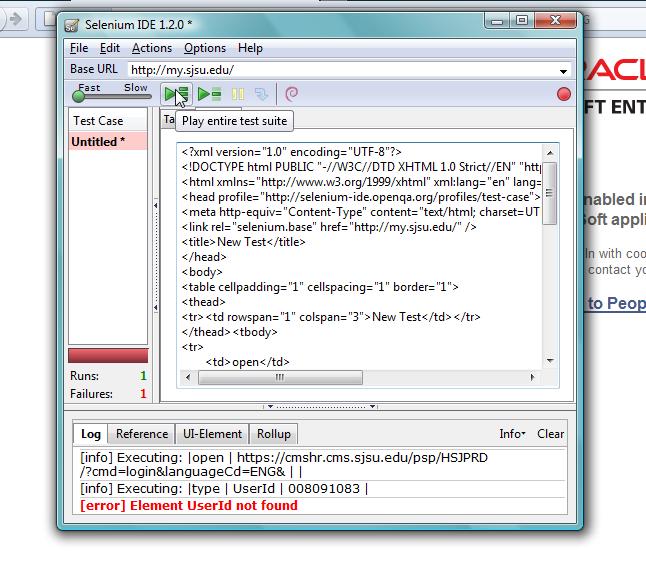


Figure 14 : Selenium Test Suite execution screenshot

We can execute the entire test suite or just the single test case.

# 7. Test Standards

## 7.1 Standards

The list of standards for performing system tests are as follows:

* + The test cases should be designed based on the system requirements specified in the design document and they should try to cover all those requirements.
  + The test cases should be properly documented, simple to understand and easy to perform.
  + The test data and test cases should have high efficiency ensuring they discover most of the defects in the functionality under test.
  + All the test cases should be traceable with the corresponding design requirement. This would prevent any missing test requirements.
  + Similar functionality tests should be designed and documented together to reduce the no. of tests and their redundancy.
  + Existing test cases should be used to match the functionality.
  + Test cases should be run independently .They should be re-run after making any changes in the code.

## 7.2 Defect Reporting

Recording/reporting the identified defects in the software program at each stage of testing process is an integral part of a successful life cycle testing approach. The purpose of this activity is to create a complete record of the discrepancies identified during testing. The information captured is useful throughout the project and forms the basis for quality measurement.

## 7.3 Defect Tracking Process



Figure 15: Bug Life Cycle

A defect tracking process should include following

### 7.3.1 Defect discovery/defect naming

Defect discovery means finding the defect, reporting the defect and acknowledging the defect if it is valid or not. Each defect log should include following details.

|  |  |
| --- | --- |
| **Description** | Short but clear description of the defect .The description should provide enough information to identify and or search the defect. |
| **Reported By** | Person reporting the defect. |
| **Date** | Date on which defect was reported. |
| **Assigned to** | Engineer being assigned to research and/or correct defect. |
| **Detected Build version** | Build version where the defect was found. |
| **Priority** | This indicates the importance or urgency of fixing the defect. Priority levels can be Urgent, Resolve ASAP, Normal queue, Low and Trivial. |
| **Severity** | This level indicates the degree of negative impact the defect has on the operation of the software application. The degree of severity can be classified as Critical, Major, Moderate, Minor, Usability, Enhancement. |
| **Fixed Build Version** | Build version in which the defect is fixed. |
| **Status** | Indicates current status of the defect like New, Open, Rejected, Fixed, Reopened , Verified , Closed etc. |
| **Defect resolution Process** | Describe the procedure used to resolve the defect. |

Table 13: Defect Details in defect log

### 7.3.2 Defect resolution

Once the developers have acknowledged that a reported defect is a valid defect the resolution process begins. This involves prioritizing the defect, scheduling the fix, removing the defect from the system and reporting resolution.

### 7.3.3 Process improvement

Improve the process and prevent potentially major failures. The process should be strengthened to prevent defects and also to find defects which have been created but not yet discovered.

# 8. Assumptions/Constraints

Testing result could be hindered or otherwise affected by the lack of any of the following items:

* + Hardware resource availability-stand alone acceptance environment will be needed.
  + Stable source control area
  + Requirement and or design exit criteria
  + Ability to mimic real time end user environment
  + Availability of tool
  + Accurate design specification document
  + Sufficient number of test personnel

# 9. Summary

Regression Testing is a type of software testing that intends to ensure that the changes either enhancements or defect fixes, to the software have not adversely affected it. This testing ensures that the changes made either to enhance the system or to fix a defect will not create new defects in the new version and the system continues to run as per the requirements. Regression testing is divided in two parts component –level and system-level. In component level each component will be tested separately. In system level testing will be performed on the whole system with all embedded components together. For this test engineer must create test cases to test system’s performance as per the requirement.

# 10. References

* SQAtester.com, Test Plan Sample,” *Documentation and Strategy*, *2006*

Website: <http://www.sqatester.com/documentation/testplansmpl.htm>

* Website: <http://softwaretestingfundamentals.com>
* Jerry Gao; Yumei Wu; Lee Chang; Sigurd Meldal; , "Measuring Component-Based Systems Using a Systematic Approach and Environment," *Service-Oriented System Engineering, 2006. SOSE '06. Second IEEE International Workshop* , vol., no., pp.121-129,Oct.2006.doi:10.1109/SOSE.2006.19  
  URL: [http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4027127&isnumber=4027102](http://ieeexplore.ieee.org.libaccess.sjlibrary.org/stamp/stamp.jsp?tp=&arnumber=4027127&isnumber=4027102)

# 11. Team Member’s Contribution

|  |  |
| --- | --- |
| **Team member’s name** | **Section** |
| Anshu Basia | Targeted testing and criteria |
| Venkata Davuluri | Testing tools, Task and schedule |
| Shubhada Narkar | Re-testing requirements, Testing standard, Assumption, Constraint, Summary, References, Appendix, Gantt Chart |
| Sonalika Gupta | Introduction, Requirement Analysis, Task and Schedule |

# Appendix A

Important Document Templates used in Testing Process.

## 1. Test case Template:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID:** | |  | **Documented Date:** |  | |
| **Test Item:** | |  | | | |
| **Tester Name:** | |  | | | |
| **Test Type:** | |  | | | |
| **Product Name:** | |  | **Version No.** |  | |
| **Required Test Scripts:** | |  | | | |
| **Operating System:** | |  | | | |
| **Comments:** | |  | | | |
| **Test Case Description:** | |  | | | |
| **Operation Procedure:** | |  | | | |
| **Pre- Conditions:** | |  | | | |
| **Post- Conditions:** | |  | | | |
| **Input Data:** | | | **Expected  Output Data:** | | |
|  | | |  | | |
| **Step:** | | | | | **Comments:** |
| 1. |  | | | |  |
| 2. |  | | | |  |
| 3. |  | | | |  |
| 4. |  | | | |  |

Detailed description of the test case template:

* Test Case ID: This is a unique identifier for each of the test cases.
* Test Item: These are the features of elevator simulation system that will be tested. (Installation, Deployment, Door and Floor Components, Performance)
* Tester Name: The author of the test case.
* Documented Date: The date on which the test case will be tested.
* Test Type: This includes which type of testing we will be following, viz. Black Box Testing or White Box Testing. In our case it is Black Box Testing.
* Product Name: The product’s name is elevator simulation system.
* Version No.: It is the next major release of elevator simulation system.
* Operating System: Operating system on which test case will be tested.
* Comments: Optional detail for describing the test scenario.
* Required Test Scripts: Name of the test script associated with the test case.
* Description: Detailed description of the test case.
* Operation Procedure: Steps required in carrying out the test case.
* Pre-Conditions: This is the initial state of the system before the test case is executed.
* Post-Conditions: The state of the system after the test case is executed.

## 2. Test case result Template

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test Case** | **Pass/Fail** | **Number of Steps** | **Steps Executed** | **Steps Passed** | **Steps Failed** | **Test Date** | **Version** | **Tester Name** | **Comments** |
| 1. |  |  |  |  |  |  |  |  |  |  |
| 2. |  |  |  |  |  |  |  |  |  |  |

## 3. Defect Reporting Format

We will use Elementool Format for defect reporting. Format is included in Tool sections screen /knowledge shots.

* + Resolved defects/Change requests
  + Training building
  + Sufficient number of test personnel