**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:**

October 12, 2011

**Submitted By**

Deepshikha Koul SJSU ID: 007534761

Bindiya Thomas SJSU ID: 007509957

Snigdha Gulhati SJSU ID: 007032519

Nimma Hemanth SJSU ID: 007508085

## TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc304389224)

[1. INTRODUCTION 5](#_Toc304389225)

[1.1. Project Overview 5](#_Toc304389226)

[1.2. Document Structure and Scope 7](#_Toc304389227)

[1.3. Project Audience 9](#_Toc304389228)

[1.4. Evaluation Criteria 9](#_Toc304389229)

[2. TARGETED TESTING AND CRITERIA 10](#_Toc304389230)

[2.1. Testing Strategy 10](#_Toc304389231)

[2.1.1. Component Testing 10](#_Toc304389232)

[2.1.2. System-Level Function Regression Testing 11](#_Toc304389233)

[2.1.3. Configuration Testing 12](#_Toc304389234)

[2.2. Testing Approach (Jerry Zeyu Gao n.d.) 12](#_Toc304389235)

[2.3. Test Criteria 16](#_Toc304389236)

[2.3.1. Black Box Phase 16](#_Toc304389237)

[2.3.1.1. Black Box Entry Criteria 16](#_Toc304389238)

[2.3.1.2. Black Box Exit Criteria 16](#_Toc304389239)

[2.3.2. White Box Phase 17](#_Toc304389240)

[2.3.2.1. White Box Entry Criteria 17](#_Toc304389241)

[2.3.2.2. White Box Exit Criteria 17](#_Toc304389242)

[2.4. Test Case Template 19](#_Toc304389243)

[2.5. Test Case Results 24](#_Toc304389244)

[3. TESTING REQUIREMENTS 28](#_Toc304389245)

[3.1. Test Environment 28](#_Toc304389246)

[3.2. Hardware Requirements 28](#_Toc304389247)

[3.3. Software Requirements 28](#_Toc304389248)

[4. TESTING TASKS AND SCHEDULE 28](#_Toc304389249)

[4.1. Roles and Responsibilities 28](#_Toc304389250)

[4.2. Project Deliverables 30](#_Toc304389251)

[4.3. Testing Schedule 30](#_Toc304389252)

[Table 4.3: Task Schedule 31](#_Toc304389253)

[4.4. Individual Testing Plan 32](#_Toc304389254)

[5. TEST TOOL USAGE AND SELECTION 35](#_Toc304389255)

[5.1. Selenium (Selenium) 35](#_Toc304389256)

[5.1.1. Selenium IDE Features (Selenium IDE, 2011) 36](#_Toc304389257)

[5.1.2. Cost & Complexity 40](#_Toc304389258)

[5.2. JUnit (www.junit.org) 41](#_Toc304389259)

[5.2.1. Features – 41](#_Toc304389260)

[5.2.2. Cost & Complexity – 44](#_Toc304389261)

[5.3. ElementTool 44](#_Toc304389262)

[5.3.1. Features: 44](#_Toc304389263)

[5.3.2. Cost and Complexity: 52](#_Toc304389264)

[6. TEST STANDARDS 53](#_Toc304389265)

[7. SUMMARY 54](#_Toc304389266)

[8. Bibliography 55](#_Toc304389267)

**Test Plan work division:**

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **Name** | **Work done in Test Plan** |
| 1. | Deepshikha Koul | * Testing Approach * Test Case template and Test Results * Test Tool – Selenium |
| 2. | Bindiya Thomas | * Test Criteria * Testing Tasks and Schedule * Summary |
| 3. | Snigdha Gulhati | * Testing Strategy * Test Tool usage – JUnit * Test Standards |
| 4. | Nimma Hemanth | * Introduction * Testing Requirements * Test Tool –Elementool |

**Team Contacts:**

Deepshikha Koul ( Deepshikha.Koul@gmail.com)

Bindiya Thomas (bindi1410@gmail.com)

Snigdha Gulhati (snigdha\_gulhati@yahoo.com)

Nimma Hemanth (hemanth228@gmail.com)

## INTRODUCTION

The test plan document aims to present a test plan which describes the testing strategies and methodologies that will be used in our testing process. The overall goal of this project is to test a component based software application – “The Elevator System”.

### Project Overview

The objective of the project is to understand the importance and benefits of component testing, system level function testing, configuration testing and regression testing for component based software applications. The present project is a component based elevator simulation system for which the above testing is to be performed. Regression testing is the selective retesting of a software system that has been modified. Regression testing ensures that the bugs are fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software. It is also named as verification testing. This type of testing is usually done when a programmer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity. Regression testing is usually performed by rerunning previously run tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. The Elevator Simulation System is developed in a component based model. The application is developed using JAVA and gives a configurable module by module deployable application to simulate.

The Elevator Systems contains the following components:

* The Admin Console Component
* The Algorithm model that drives the entire system
* The car as a whole Component
* The car controller Component
* The door and its operational Components
* The door panel Components
* The Floor panel Components
* The meta controller Component
* The user panel Component
* The user panel queue Component
* Floor Indicator

As part of the new functional requirements for the project, the following new components will be added to the Elevator System:

1. Internal Alarm Component
2. Emergency Alarm

A new Elevator Algorithm will be chosen.

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Component to be added/ Changed** | **Java class to be changed** |
| Changing the Algorithm Component | - Algorithm  - Metacontroller | - Introduce new algorithm class : LRU.java  - AlgorithmFactory.java  - MetaController component files |
| Addition of Alarm component | - AdminConsole  - FloorPanel  - CarController  - UserPanel  -Car | - GUI.java  - UserPanelUI.java  -FloorPanelUI.java  -Introduce new : AlarmStatus.java  - CarController |

**Details: Changing the Algorithm Component**

The algorithm that will be introduced will be LRU ( Least Recently used Algorithm ).  LRU discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards *the* least recently used item.

**Details: Addition of Alarm Component**

The Addition of Alarm component will involve changes in GUI. This will involve having an alarm button in the user panel , which will be in OFF status. Once this Alarm Button is pressed the color of the button will change from green to Red and the car will come to a stop. Simultaneously, the status of the car being stuck between floors will be highlighted on the floor panel.

The above components would be developed by the Team Developer. Black box and White box testing will be performed on the newly added components. Black box testing methods will be used to carry out System – Level function regression testing to identify component based software changes and impacts.



Figure 1.1 Component Composition Diagram

### Document Structure and Scope

The Regression Test Plan document of the Elevator System aims to find out new errors in the already existing functionalities of the Elevator System due to the updates or modifications made to the system. The Test Plan document is the initial phase of the Software Testing Lifecycle. It describes the objective of the project, the testing approach selected in order to carry out regression testing. It lists out the functionalities which are to be tested; the scope of testing, the roles and responsibilities of each member of the team are well defined. It specifies the schedule of the entire testing life cycle, the resources that will be used, deliverables, risks and environment. The document also contains information about the various testing and management tools that will be used to test the application and to log and track the errors/bugs found in the system. The overall purpose of the test plan document is to describe the project and the sequence of steps that will be executed to perform regression testing of the component based elevator system.

**Scope:**

In component testing only two components are selected from the Elevator System and testing is performed on them, the components chosen are:

1. Floor indicator component
2. User Panel

To implement Black Box testing two specific test design methods will be identified along with appropriate test coverage. The methods chosen are:

* 1. Equivalence Partition.
  2. Decision Table Testing

To implement White Box testing two specific test design methods will be identified along with the appropriate test coverage. The methods chosen are:

1. Data Flow Software Testing
2. Basis Path Testing

The methods that will be used to carry out system-level regression testing are:

1. Black box testing method
2. Equivalence partition
3. Decision table

In Regression Testing the code change will be performed only at the specified components.

Test complexity and cost will be measured for only System-Level Function Testing and Regression Testing.

### Project Audience

The Project audiences are the set of people who are related or involved in the project being developed or being executed.

They include:

1. Professor Jerry Gao (Project Guide)
2. Members of our team (Test manager, Test engineers ,Developer and Regression Test Engineer)

### Evaluation Criteria

The test cases should be carefully generated such that it can be traced back to a specific test requirement. The test cases written should be covering the overall scope of the project and should be made sure that no requirements are left. The requirement traceability matrix is designed for such a requirement; this avoids the risk of missing any test requirements. . The new components that are being added to the Elevator System are integrated with the existing components. Due to the addition of these new components the other components of the Elevator System may be affected and they should also be tested. For re-testing these components the existing test cases can be made use of and new test cases can be added if required or existing test cases can be removed if not required. The test cases are then executed either manually or by automation and are checked with the expected results to classify them either into pass or fail. The results from each test case are noted and are documented for further assistance. The defects that are raised due to testing would be logged in a defect tracking tool and its reproducible steps would be logged in it. Then the defects would be reported to the testing lead or manager for further actions to be taken.

## TARGETED TESTING AND CRITERIA

### Testing Strategy

In order to extensively test the changes made to the Elevator System as part of this project, our testing strategy would include 3 types of testing -

### Component Testing

Component Testing involves testing of individual components to make sure that the changes made are correct and as per the requirements specifications. This testing phase would involve both Black box and White box testing using 2 test design methods each. We would test one new component and one modified component.

***Black box Testing –*** Black box testing is used to test the functionality of the system based on the requirements and functional specifications. The Black box test design methods we would be using in this project are –

* **Decision table-based testing –**

Decision Tables are an elegant way to associate conditions with actions while testing any system. Various conditions and actions are identified based on the requirement specifications and corresponding test cases are derived.

**Coverage:** Using Decision Table testing, all the business rules and constraints of the system will be validated. All possible conditions and their resulting actions or outcomes would be tested. For each combination of conditions and actions, a test case would be formulated.

* **Equivalence Partitioning –**

Equivalence Partitioning testing method involves dividing the input domain into various disjoint partitions. Each equivalence class represents both valid and invalid set of inputs.

**Coverage:** Functionality of the component would be tested by randomly selecting one input value from each partition. Test cases would be generated to test for both valid and invalid data.

***White Box Testing –*** White box testing is used to test the logic of the component and test for internal and programming errors. It requires working knowledge of the system, unlike Black box testing. The White Box testing methods we would be using in this project are –

1. **Data Flow Software Testing**

In Data Flow Software testing, the major focus is on the data. This testing focuses on the data flow definition and its uses in the control flow charts. It focuses on the entire lifecycle of a piece of data, thus testing for all possibilities of bugs.

**Coverage:** Because the entire lifecycle of the data is tested, Data Flow testing provides improved data test coverage of the software.

1. **Basis Path testing**

In Basis path testing, various execution paths are defined and the tests are based on these execution paths.

**Coverage:** Basis Path testing guarantees executing every statement in a program atleast once. It takes into account the Statement coverage, Branch coverage, Multiple condition coverage, path coverage and Loop coverage.

### System-Level Function Regression Testing

System Level Function Regression testing is to test the entire system, and make sure that there is no negative impact on the system as a whole after adding new or modifying existing components. We would be performing Black box testing to ensure proper working of the system.

We would be using the same Black box test methods as Component testing, which are –

* Decision table-based testing
* Equivalence Partitioning

**Coverage:** The system functionality, as a whole would be tested, without considering individual components.

### Configuration Testing

Configuration testing is the technique to test the various configurations and ensure that the system works fine with all possible configurations.

We would be testing two configurations. The test method is to be decided.

### Testing Approach (Jerry Zeyu Gao n.d.)

Software testing is an essential part of SDLC (Software Development Life cycle). The main objective of Software testing is to enhance the quality of the product delivered, in our case being the Elevator Software system. The essential elements in software testing include the source code, design patterns and details, component and system requirement specifications , testing approach - unit testing , integration testing , validation testing and system testing. The amalgamation of all these processes results into a testing strategy.

The Project approach that we will follow has been highlighted below:

Project Planning

New Component Specifications

Design / Build New Component

Design/ Build System Test

Design / Build System Test Procedures

Building Test Environment

Executing System Tests

Bug Reporting and Management

Develop New Component

Figure 2.1: Process Flow

The diagram above outlines the Project approach that will be followed.

1. **Project Planning :** This involves creating a project plan , schedule , roles and responsibility, approach that will be followed and monitored during the project.
2. **New Component Specifications:** This involves defining the requirement specifications for the components that will be added to the elevator system.
3. **Design / Build New Component**: This involves
4. **Develop New Component**: This will lead to coding the new component.
5. **Design / Build System Test:** This will involve generation of test cases, identifying entrance, exit criteria, coverage and expected results. The output of this process uses the system specifications to generate test conditions.
6. **Design/ Build System Test:** This involves setting up procedures and standards for status tracking, error handling or bug management.
7. **Build Test Environment:** This involves setting up the right infrastructure for testing the system. This includes hardware, software and data setups.
8. **Execute System Tests**: This includes implementing the system tests and documenting the results.
9. **Bug Reporting and Management**: This involves identifying the issues with the system and reporting it to the development team who can fix the issue. Only if the pre-defined exit criteria is accomplished the process flow is completed.

In our Elevator System, the new components that will be added are

1. Internal Alarm Component: This component will include switch that can be turned on and off by the user.
2. Emergency alarm : Each Floor will have an emergency alarm ( external alarm )

Our revised Elevator system will have a 2 model alarm system.

Along with this the components that will be modified in the existing Elevator system include

1. Floor indicator
2. Floor Panel
3. User Panel
4. Elevator Algorithm

The flow diagram below shows the process that will be implemented in developing and testing the alarm component.

Alarm component Planning

Alarm component Specifications

Alarm component Development

Alarm Component test Planning

Alarm Component test Specifications

Alarm Component test Execution

Alarm Component test Recording

Alarm Component test Completion

Figure 2.2: Alarm Component development and testing workflow

### Test Criteria

Test criteria include the benchmarks or standards against which test methods and their outcomes are compared. Every test must have an accepted set of entry criteria before the testing begins and a set of exit criteria that must be met before the test is deemed as successful.

The following section describes entry and exit criteria for White box and Black box testing methods:

### Black Box Phase

* + - 1. **Black Box Entry Criteria**

The Black Box entry criteria are dependent on the system’s specifications and the user requirements. A functional system is a good criterion to start with.

* + - 1. **Black Box Exit Criteria**

Since Black Box testing tests the different functional aspects of a system from the user’s perspective, it is difficult to achieve 100% coverage. This is because all the different user scenarios cannot be tested out before the system testing is deemed complete. However, high coverage would be a good exit criterion. Some of the different methods that can help achieve adequate coverage for Black Box testing include:

* **Equivalence Partitioning:**

Equivalence partitioning divides the entire input domain into partitions. Test cases can be then derived from these partitions. By dividing the entire input domain and deriving test cases from them, this method helps to reduce the number of test cases that must be run to adequately test the system. The different partitions are derived based on the requirement specification for the input values.

* **Boundary Value Analysis**

This method focuses on testing the system response to boundary values of the inputs specified in the requirements document. This method splits the input domain into different boundaries and test cases are designed around the values at the boundary. By focusing on boundary values, this method uncovers many errors as the possibility of errors near the boundary values is high, thus helping to achieve adequate coverage.

* **Decision Table**

Decision table testing is another method that helps to increase coverage in Black Box testing. This testing method helps to analyze complex scenarios where combinations of conditions are required to produce certain specific output/action. The basic approach of the method is to identify the different conditions in the system and the actions for a combination of those conditions. Equivalence partitioning and boundary value analysis assumes that all the variables in the test are mutually independent. However this is not the case in real world scenarios. Decision table method entails creation of a test case for each combination of conditions.

The above mentioned methods will help to adequately test the functionality of a system thus helping to achieve high coverage. The high coverage can be a sufficient test exit criteria.

* + 1. **White Box Phase**

White Box testing tests the internal methods or workings of an application by focusing on the program’s logic, internal data structures and functions. It is also known as glass box, clear box or transparent testing. For designing test cases for white box testing, a deep understanding of the internal perspective of a system is needed.

* + - 1. **White Box Entry Criteria**

White box testing tests the internal logic of a component or module. The entry criteria for white box testing include compliable source code that can be run independently.

* + - 1. **White Box Exit Criteria**

Since white box testing is dependent on the source code, we can adequately measure the test coverage. High test coverage, typically greater than 90% would be sufficient to exit the white Box Test Phase. To achieve this high coverage it is imperative to achieve the following:

***Basis Path Testing***

* **Statement Coverage:**

Statement Coverage can be achieved by executing every line of the program at least once. High statement coverage can help eliminate dead code. Though is a basic requirement for high coverage, it is very difficult to find a test suite that achieves adequate coverage.

* **Branch Coverage:**

Branch testing ensures that all the branches of a condition are tested for the right behavior. This is also essential to achieve high code coverage.

* **Path coverage:**

Path coverage is needed to ensure that all the elementary paths of the code have been tested at least once.

* **Loop Coverage:**

In loop testing it is important to check all the boundary value conditions for loops just as is done in black box testing. Loop coverage is also essential to achieve high code coverage.

***Data Flow testing Coverage:***

It is imperative to include sufficient data flow coverage when discussing white box test exit criteria. Data flow testing is testing by manipulating data, which is central to the program. The testing is done for abnormal situations that can occur with data, which include:

* Variable being used before it is defined
* Variable being defined but not used
* Variable being defined twice before it is used

Data Flow testing adequacy is guaranteed by the following basic criterion:

* All-defs: This criterion is satisfied only if there is a clear path from a node in which Variable v is defined to a node which has either a computational use(c-use) or predicate use(p-use) of v.
* All-uses: This criterion is satisfied only if there is a clear path from every node that has a definition of variable v, to every node that has computational uses and to every node that has predicate uses of v. This criterion can be further subdivided into 4 categories: all c-uses/some p-uses, all p-uses/some c-uses, all c-use and all p-use.

### Test Case Template

We will be using Elementool to manage and monitor the testing of Elevator System Project. In Elementool (Elementool), the test cases can be linked to the issues found while testing the project. Also the status of the issues associated with each test case can be monitored individually though the tool. The test case template provided by Elementool is used to document the test cases as shown below:

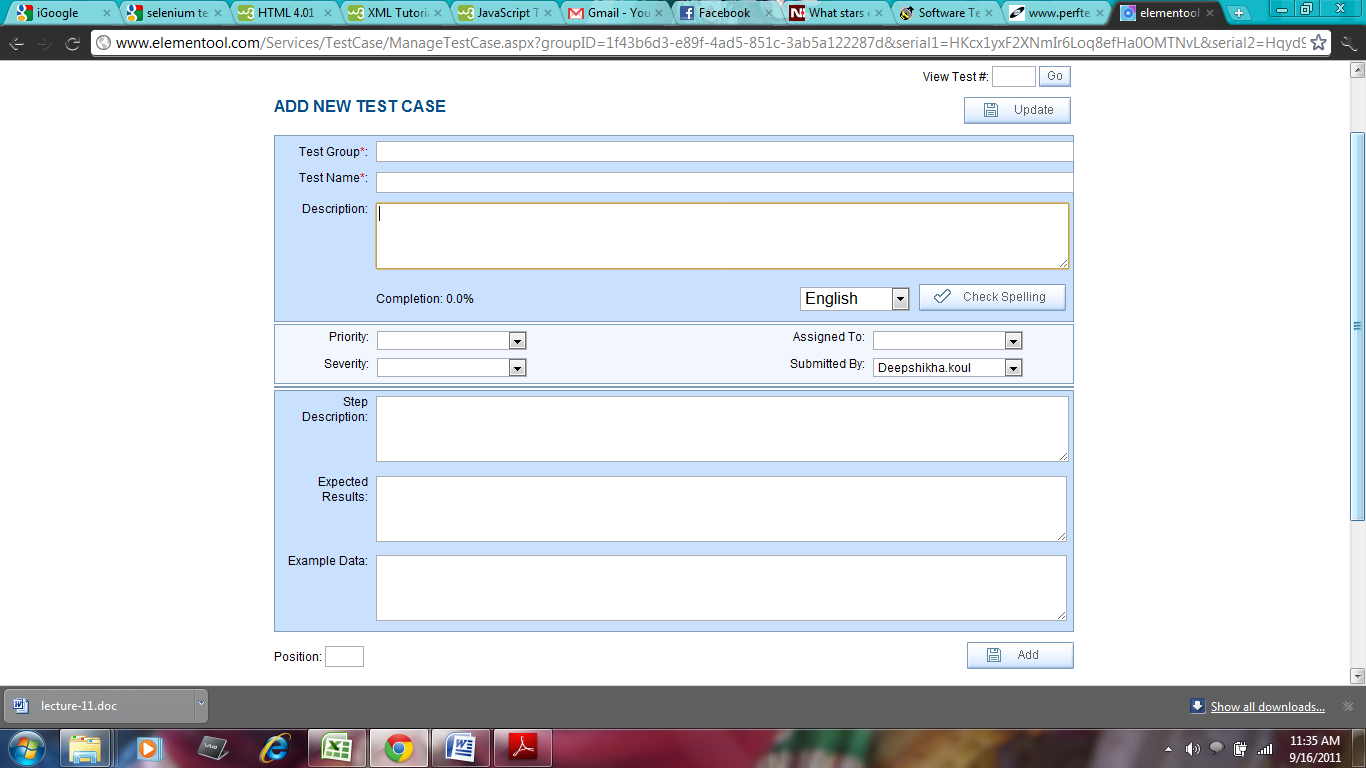


Figure 2.3 : Element tool Test Case Screen

The information captured in this section is as follows:

1. Test Group: These are the features of elevator simulation system that will be tested. (Installation, Deployment, Door and Floor Components, Performance)
2. Test Name: Name of the Test case or the test that will be performed.
3. Description: This will have the detailed description of the test case and will include
4. Test Type: White Box Testing / Black Box Testing
5. Testing Environment: The Operating system and the system conditions based on which the test case will be executed.
6. Details describing the test scenario
7. Pre-Condition: Initial state of the system before the test case is executed.
8. Post Condition: Final state of system after the test case is executed.
9. Priority: Priority of the test case - Critical / High / Medium / Low
10. Severity: The Severity of the test case - Critical / High / Medium / Low. This is essential for risk estimation.
11. Assigned to: The test case is assigned to a Test engineer.
12. Submitted by: The author of the test case
13. Step Description: This details out systematic description of the test. These steps define the order of execution of the test.
14. Expected Results: Expected results for the step defined in the Step description
15. Example data: Sample data for the step as defined by the description

Once the test case is added to the tool, below mentioned fields are added to the test case:

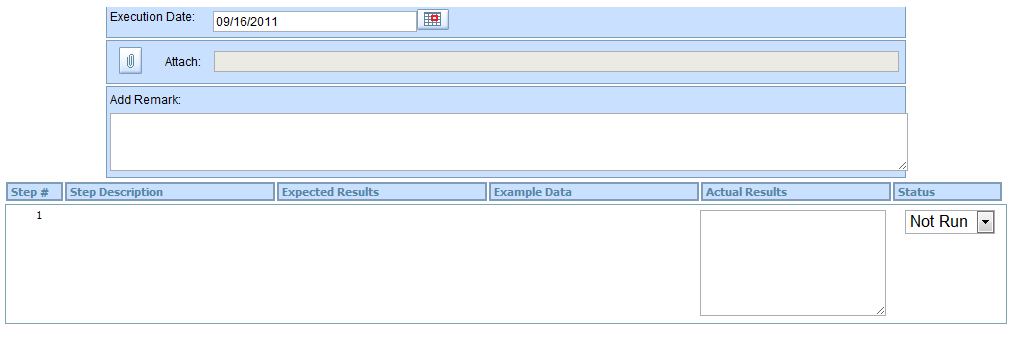
1. Test Case #: A unique identifier for each test case will be auto populated by tool.
2. Execution Date: The date on which the test case will be executed and tested. As a default, the tool will auto-populate the date on which the test case was generated. This date can be changed based on the project schedule.
3. Status: After the test has been performed the Status - Not Run /Pass / Fail for the step is updated by the Tester.
4. 

Figure 2.4 : Step Description and Actual Result Update screen

1. Actual Results: The result found after the test is executed.
2. Linked Issues: The issues/ bugs found while testing , can be linked to the test case by below mentioned screen

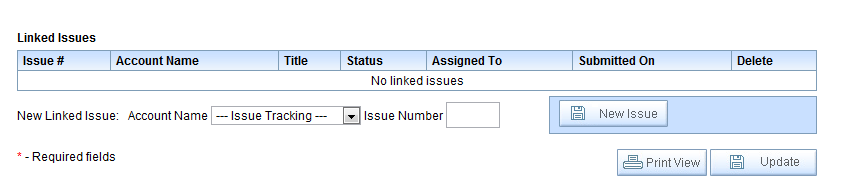
****

Figure 2.5: Linked Issues Screen

In case the Elementool is not available, the following template can be used to document the test case

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Template** | | | | | | | | |
| Test Case # |  | Test Name | |  | | Test Type |  | |
| Priority |  | Tester Name | |  | | Test Date |  | |
| Description |  | | | | | | | | |
| Required Test Scripts |  | | | | | | | | |
| Pre-Condition |  | | | | | | | | |
| Post-Condition |  | | | | | | | | |
| Input Data |  | | | | | | | | |
| ***Testing Environment Details: (If multiple testing environments are available, repeat the below table.)*** | | | | | | | | | |
| Machine  Name |  | DB Directory | |  | | Operating System |  | |
| Version |  | DB | |  | | Client Server /Back-End |  | |
| ***Testing Scenario Details :*** | | | | | | | | |
| ***TC Step*** | ***Step Description*** | | ***Expected Results*** | | ***Actual Result*** | | | ***Pass/Fail*** |
|  |  | |  | |  | | |  |
|  |  | |  | |  | | |  |
|  |  | |  | |  | | |  |
|  |  | |  | |  | | |  |
| **Comments** |  | | | | **Test Case Status** | | |  |

Table 2.1: Test Case Template

### Test Case Results

The results of the test case execution are recorded using the Elementool (Elementool). Any issue or a bug found while executing the test cases are reported immediately using Elementool. The following screenshot shows the test case results and the status of each

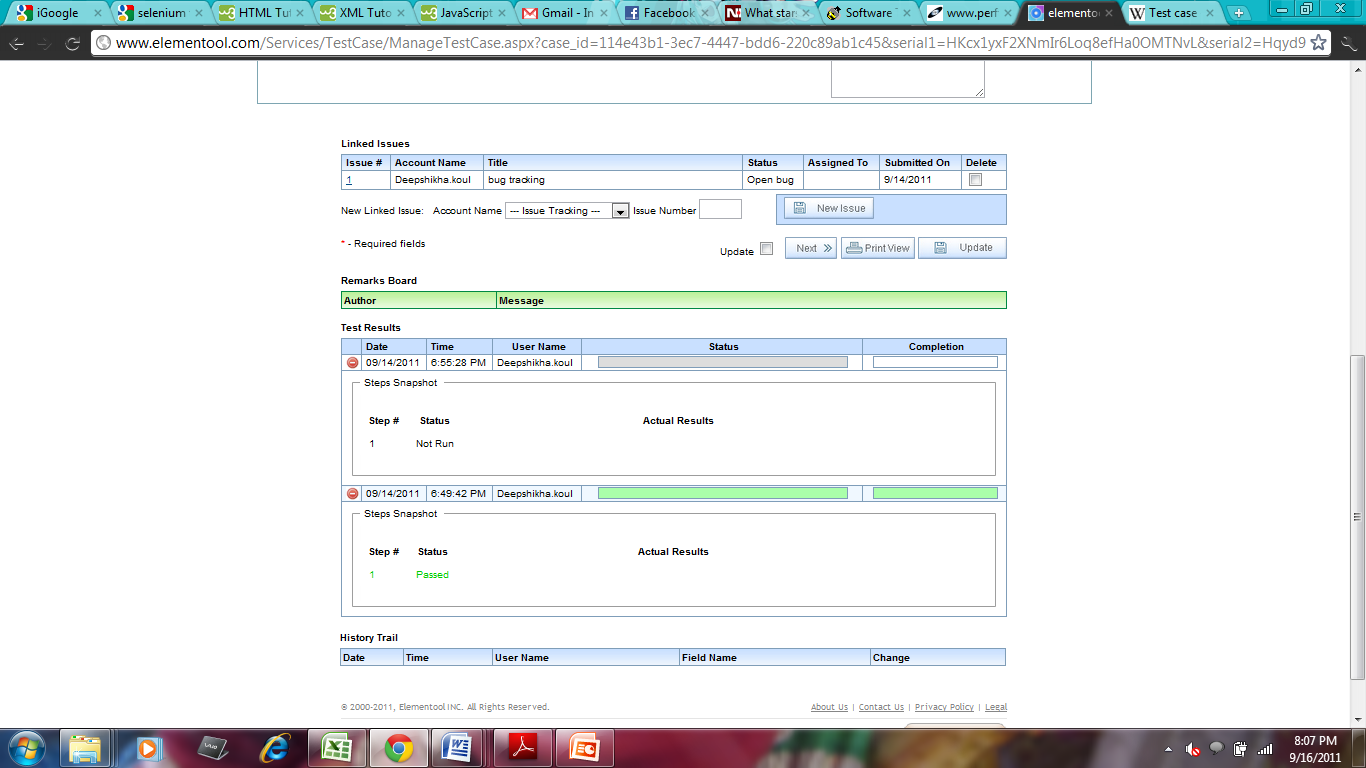
associated step.  


Figure 2.6 : Test Case Results

In order to document any issues associated with the test case, tester needs to click on NEW ISSUE and the below highlighted screen will be generated.

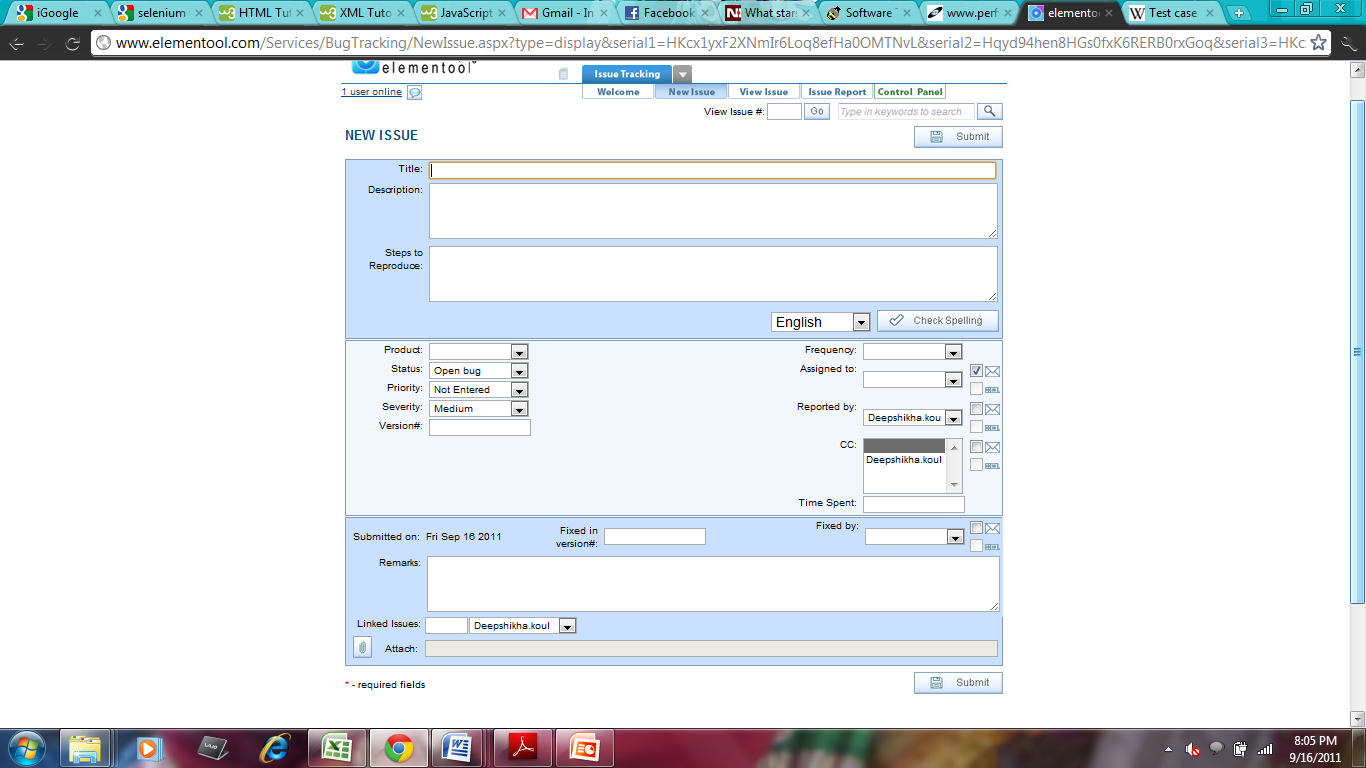


Figure 2.7 : New Issue Screen

The information captured in this section is as follows:

1. Title: The name of the issue / bug / problem
2. Description: This will have the detailed description of the issue and will cover the environment in which the bug occurs.
3. Steps to reproduce: Detailed systematic procedure to generate the bug again
4. Product:
5. Status: Open / Closed / Fixed / Implemented / Duplicate
6. Priority: Immediate / High / Medium / Low
7. Severity: Critical / High / Medium / Low
8. Version #: The version of the software being tested
9. Frequency: Frequency of the issue generation - Sometimes / Always / Often / Randomly
10. Assigned To: Developer / Manager to whom the issue is assigned.
11. Reported By: Tester who found the issue
12. Remarks: Optional findings
13. Linked Issue : Associated issues
14. Time Spent : Time spent to find the issue
15. Fixed in Version #: The version of the system in which the issue is rectified
16. Fixed By: The Developer who fixed the issue

In case, Elementool is not available we will use the following template to document the test case results.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Result Template** | | | | | | | | |
| Test Type |  | Test Date & Time | |  | | Tester Name |  | |
| Description |  | | | | | | | | |
| Component Name |  | | | | | | | | |
| ***Testing Scenario Details :*** | | | | | | | | |
| ***TC Step*** | ***Test Case*** | | ***Step Executed*** | | ***Status ( Pass / Fail)*** | | | ***Comments*** |
|  |  | |  | |  | | |  |
|  |  | |  | |  | | |  |
| **Issues** |  | | | | **Total Number of Steps** | | |  |
| **Steps Passed** |  | | | | **Steps Failed** | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Result Template** | | | | | | | | | | | | |
| **Testing Type** | | | | | **Product Name** | | | **Description** | | | | |
|  | | | | |  | | |  | | | | |
| **No.** | **Test Case** | **Pass/Fail** | **Number of Steps** | **Steps Executed** | | **Steps Passed** | **Steps Failed** | | **Test Date** | **Version** | **Tester Name** | **Comments** |
| 1. |  |  |  |  | |  |  | |  |  |  |  |
| 2. |  |  |  |  | |  |  | |  |  |  |  |

Table 2.2: Test Case Result Template

## TESTING REQUIREMENTS

### Test Environment

A test environment can be thought of has a setup of software and hardware on which the testing of the Elevator System is performed. The Test Environment consists of the physical setup which includes the hardware, and logical setup that includes the client operating system, JDK 5.0, eclipse IDE setup and other required software for testing.

### Hardware Requirements

The hardware requirements for testing the different components in the Elevator System are:

* Intel Core 2 Duo based PC 1.5 GHZ Processor
* 1 GB RAM
* 10 GB hard drive space

### Software Requirements

The software requirements for testing the different components in the Elevator System are:

* Schedule: Microsoft Office 2007
* IDE: Eclipse IDE for Java with JDK 5.0 or later version

Operating Systems: Windows XP/VISTA/7, Linux

Testing Tools: Selenium, Junit, Elementool

## TESTING TASKS AND SCHEDULE

### Roles and Responsibilities

|  |  |
| --- | --- |
| **Roles** | **Responsibilities** |
| **Test Manager**  **(Deepshikha)** | * Coordinate testing efforts of the team * Establish standards and processes for testing * Establishing test scope * Creating testing plan * Select test methods and models * Create detailed testing schedule * Assign project tasks to the team members * Selecting test tools * Perform white box and black box testing * Perform System Testing * Conduct Configuration Testing |
| **Test Engineers**  **(Bindiya and Snigdha)** | * Analyze requirements * Participate in test Planning * Writing black box test cases for new alarm component * Writing black box test cases for new elevator algorithm * Prepare test case document * Deciding test input and output * Documenting test data * Perform Component Testing * Conduct system testing based on test cases and record the data * Performing configuration Testing * Reporting bugs * Preparing test results document. |
| **Developer**  **(Hemanth)** | * Understand the source code * Update the code to add alarm component and change other affected components like user panel and floor panel. * Modify code to change algorithm * Conduct white box testing * Prepare white box test document * Update the regression test cases to test the new functionality * Perform white box testing * Perform System level testing * Perform Configuration Testing |

Table 4.1: Roles and Responsibilities

### Project Deliverables

|  |  |
| --- | --- |
| **Milestone** | **Date** |
| **Regression Test Plan Document** | 9/21/2011 |
| **Regression Test Design Specification** | 10/19/2011 |
| **Manual Regression Test Document** | 11/16/2011 |
| **Configuration Test and Complexity Analysis Report** | 11/30/2011 |
| **Automatic Regression Testing Report** | 12/07/2011 |
| **Project Demo** | 12/07/2011 |

Table 4.2: Project Deliverables

### Testing Schedule

Preparing the testing schedule is one of the most important project activities. A good, well thought out schedule will help to ensure the successful completion of the project. Understanding the requirements of the project is essential to prepare a good schedule. We start out with understanding the system functionality, what we have to accomplish and testing the system out comprehensively to ensure that the final product is of the highest standards. The table below describes the testing schedule the team plans to follow:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Start Date** | **End Date** | **Duration** |
| **Regression Test Plan Document [First draft]** | 9/9/11 | 9/14/11 | 5 days |
| **Regression Test Plan Document [Final draft]** | 9/15/11 | 9/21/11 | 6 days |
| **Regression Test Design Specification [First draft]** | 9/29/11 | 10/3/11 | 4 days |
| **Regression Test Design Specification [Final draft]** | 10/6/11 | 10/18/11 | 12 days |
| **Manual Regression Test Document [First draft]** | 10/20/11 | 11/6/11 | 17 days |
| **Manual Regression Test Document [Final draft]** | 11/8/11 | 11/16/11 | 8 days |
| **Configuration Test and Complexity Analysis Report[First draft]** | 11/18/11 | 11/22/11 | 4 days |
| **Configuration Test and Complexity Analysis Report[Final draft]** | 11/22/11 | 11/26/11 | 4 days |
| **Automatic Regression Testing Report[First draft]** | 11/26/11 | 11/30/11 | 4 days |
| **Automatic Regression Testing Report[Final draft]** | 12/1/11 | 12/6/11 | 5 days |
| **Project Demo** | 12/01/11 | 12/7/11 | 6 days |

**Table 4.3: Task Schedule**

### Individual Testing Plan

|  |  |  |
| --- | --- | --- |
| **Team** | **Task** | **Time** |
| Bindiya | * Getting acquainted with the elevator system * Designing black box test cases for the alarm component and other modified components as well as checking the elevator algorithm * Designing white box test cases for the alarm component and other modified components * Executing the regression test cases for the elevator system * Executing test cases for the newly added alarm system * Executing test cases to check the modified algorithm * Conducting system level testing * Conduct Configuration Testing * Preparing the test report | 9/1/11-9/18/11  9/22/11-10/11/11  10/11/11 -10/18/11  10/20/11-10/28/11  10/28/11-11/6/11  11/6/11-11/16/11  11/19/11-11/25/11  11/25/11- 11/30/11  11/26/11-12/6/11 |
| Hemanth | * Getting acquainted with the elevator system * Understanding the source code * Updating the elevator system with the alarm component * Modifying the elevator algorithm and other modifiable components like floor panel indicator, user panel etc. * Design white box test cases for the alarm component and other modified components * Execute test cases for the alarm component and other modified components * Perform system level testing for the elevator system * Conduct Configuration Testing | 9/1/11-9/18/11  9/20/11-9/25/11  9/25/11-10/3/11  10/3/11-10/10/11  10/10/11-10/18/11  10/21/11-11/15/11  11/21/11-11/25/11  11/25/11 -11/30/11 |
| Deepshikha | * Getting acquainted with the elevator system * Designing Black Box test cases for the alarm component and other modified components * Designing white box test cases for the alarm component and updated elevator algorithm * Executing black box and white box test cases for the alarm component and other modified components * Designing system level test cases for the elevator system * Executing system level test cases for the alarm component. * Preparing the test report document | 9/1/11-9/18/11  9/22/11-10/1/11  10/1/11-10/11/11  10/15/11-11/06/11  11/6/11-11/15/11  11/15/11-11/30/11  12/1/11-12/7/11 |
| Snigdha | * Getting acquainted with the Elevator system * Deigning white box and black box test cases for the alarm component and other modified components * Designing Black box test cases for the modified elevator algorithm * Executing white box test cases at component level * Executing black box test cases at component level * Executing system level test cases * Conduct Configuration Testing * Preparing test report document | 9/1/11-9/18/11  9/22/11-10/11/11  10/11/11-10/19/11  10/19/11-11/8/11  11/8/11-11/13/11  11/13/11-11/25/11  11/25/11 -11/30/11  12/1/11-12/7/11 |

Table 4.4: Individual Testing Plan

## TEST TOOL USAGE AND SELECTION

### Selenium (Selenium)

Selenium is a tool for automating web applications for testing purposes. It provides a robust framework for testing web applications. It has a JavaScript framework that runs in web browsers. Selenium can work on entities, which support JavaScript. In order to execute tests, Selenium provides a record / playback functionality. These recorded tests, simulate user navigation through the web pages. In addition it provides Selenese which is a test domain specific language to write tests in various programming languages like C#, Java , Groovy , Perl , PHP, Python and Ruby. Selenium is an open source tool and can be downloaded without a charge.

Various components of Selenium include the following

* **Selenium IDE:** IDE (Integrated Development Environment) for Selenium tests This IDE is implemented as a Firefox plug-in and it helps in recording test cases. These test cases can be edited and debugged using Selenium IDE. The test scripts can be edited manually. Clicking a link , selecting an option are few of commands that can be executed using Selenium IDE.
* **Selenium Remote Control (RC):** This is a server written in java which accepts test commands for the web browser through HTTP. This provides an option to write automated tests in different programming languages using client drivers for Java, Perl , .NET,PHP, Python and Ruby. This ensures better integration of Selenium in existing unit test frameworks.
* **Selenium Web driver:** It accepts commands sent via Selenese language or through Client API using a browser specific driver and sends these commands to the browser in order to retrieve results. Selenium Web driver does not require a special server to execute tests. It starts a web browser instance and controls it.

**Selenium Grid: Selenium Grid** isa server that allows executing parallel tests by using web browser instances running on remote machines. One server acts as a hub that provides access to browser instances. The browser instances are also known as Web Driver nodes. Thus, the tests are executed on these Web driver nodes through the Hub.

For testing our Elevator project, we will be using Selenium IDE.

### Selenium IDE Features (Selenium IDE, 2011)

**Starting Selenium IDE in Firefox**

Start Selenium IDE in Firefox by going to Tools>Selenium IDE. The below highlighted screen shot will be generated.

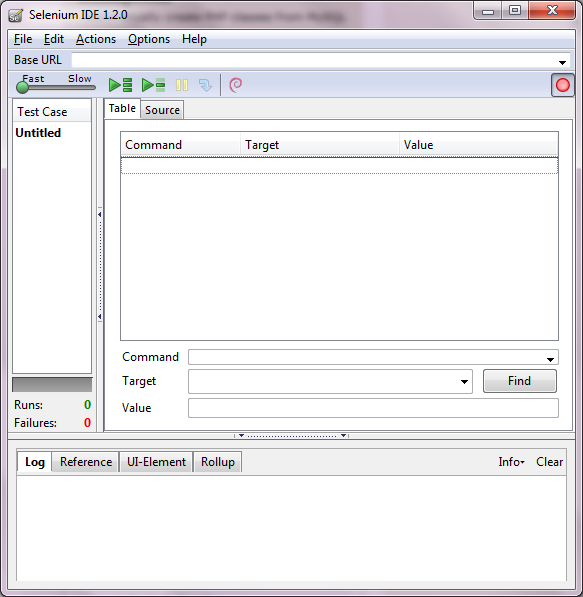
****

Figure 5.1 : Selenium IDE screen

The various features of Selenium IDE have been highlighted below.

* **Selenium Menu Bar**

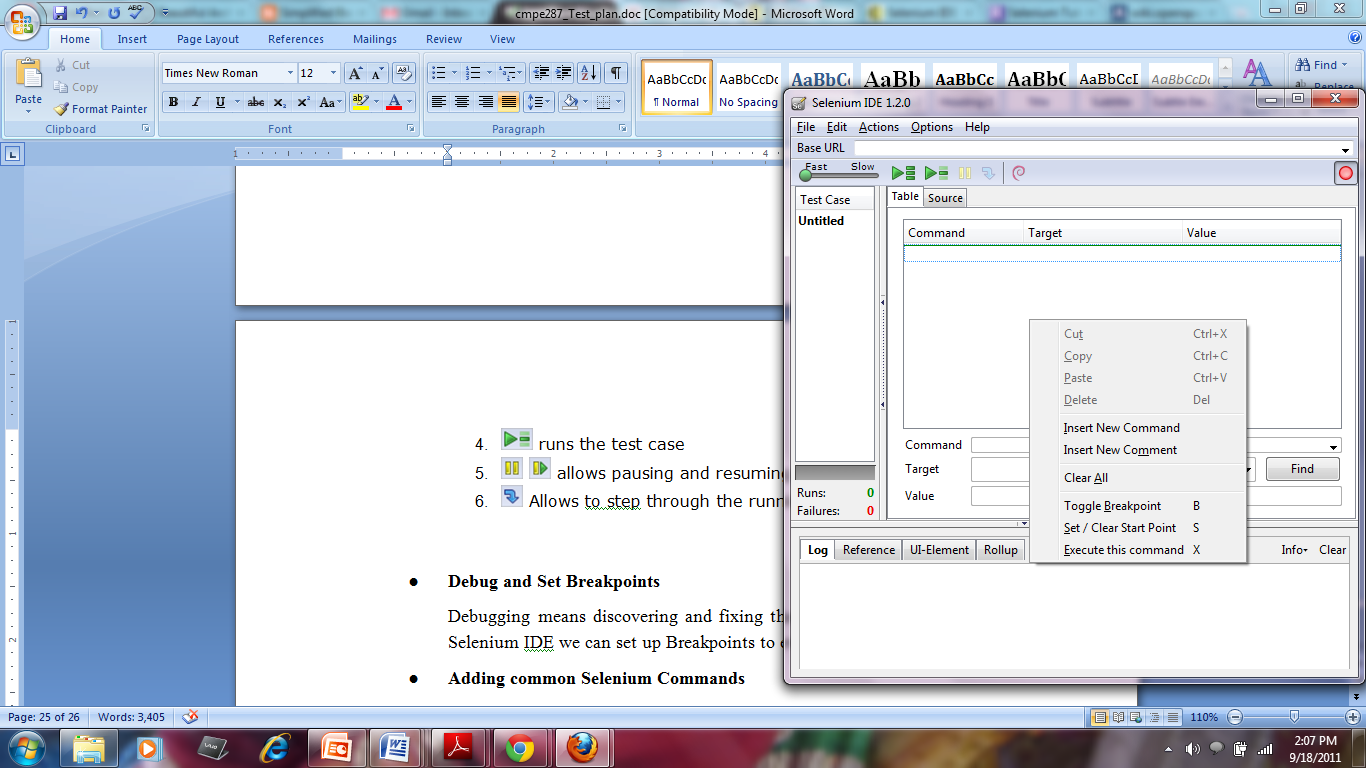


Figure 5.2 : Selenium Menu Bar

The menu bar allows opening, creating, saving and exporting test cases and test suites through the File option. Edit menu provides options for editing the commands in the test case.

* **Record and playback the test case**

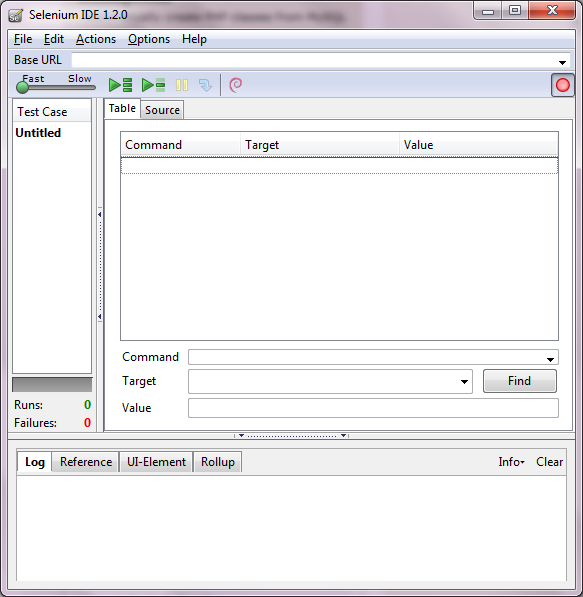
****

Figure 5.3 : Selenium Toolbar

1. By Clicking in the Red button, we can record the user browser actions
2. **speed control** Controls the speed of running the test case
3. **run all** runs the entire test suite
4. run runs the test case
5. pause resume allows pausing and resuming the test case
6. step Allows to step through the running test case.

* **Debugging through Breakpoints and start points**

Debugging means discovering and fixing the errors in the test cases. Using Selenium IDE we can set up Breakpoints to evaluate the test case. Breakpoints can be set up by right clicking on the command and selecting Toggle Breakpoint. Similarly start points can also be set up by right clicking on the command screen as shown in the below screenshot.

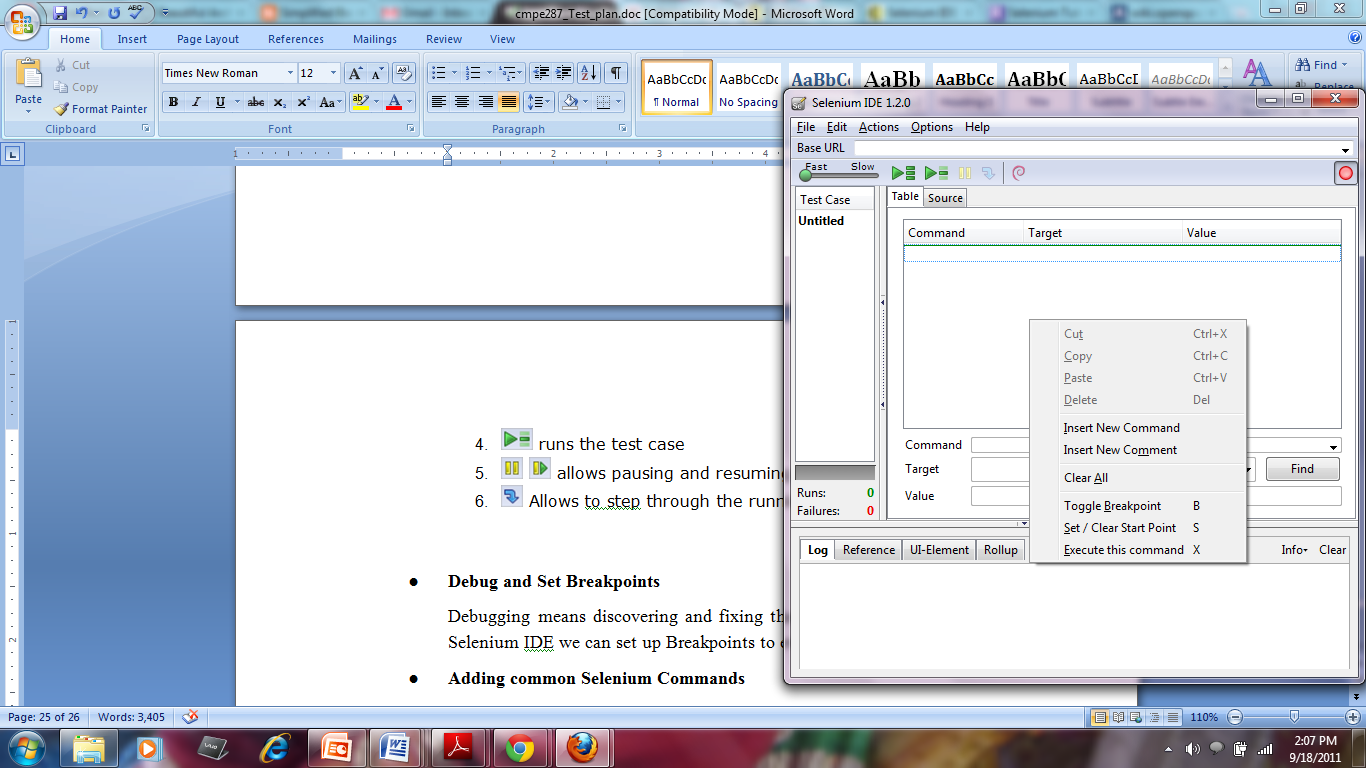


Figure 5.4: Setting up Start Point and Breakpoint in Selenium IDE

* **Test Case Pane**

This pane displays the test script that is being executed. As shown in below figure, it has 2 tabs - Table and Source. Source displays the test case in HTML format and Table displays the command, parameter values and the target information.

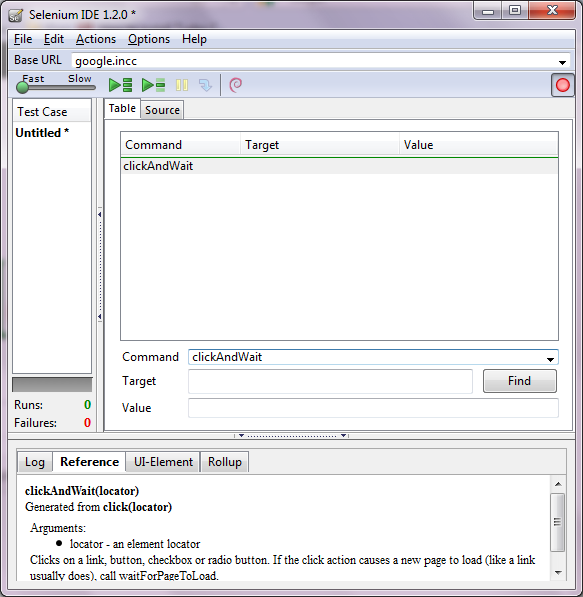


Figure 5.5 : Test Case Pane

* **Log / Reference Pane**

Log pane reflects the progress and error information when a test case is run. This information helps in the debugging process. Clear button can be used to clear out the log.

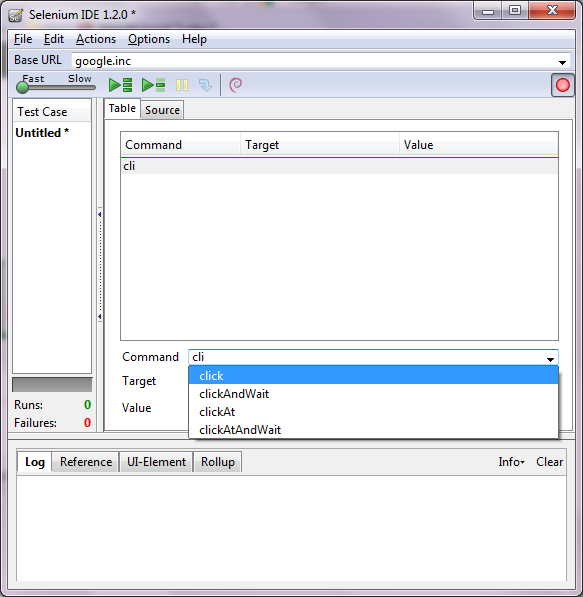


Figure 5.6 : Log pane

Reference tab provides details about the Selenese commands and arguments in these commands.

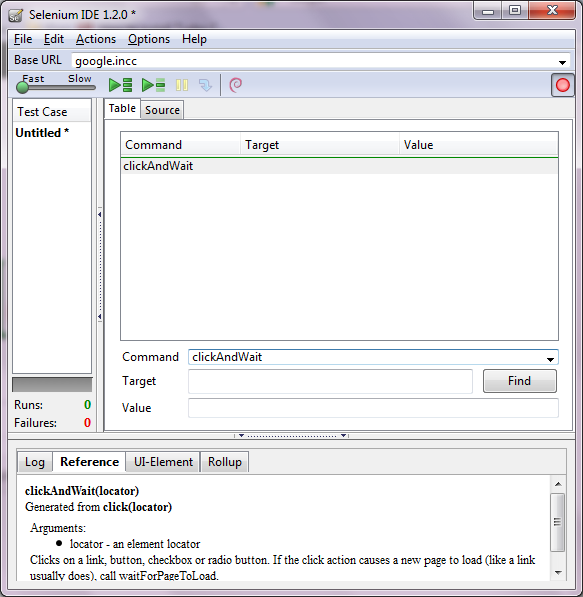


Figure 5.7: Reference Pane

* **Executing common Selenium Commands**

Selenium commands are a set of commands that run the tests. A test script is generated by a sequence of these commands. There are three main types of commands in Selenium - Actions, Accessors and Assertions. Selenium IDE provides auto complete feature for all of its commands as shown in the screenshot below

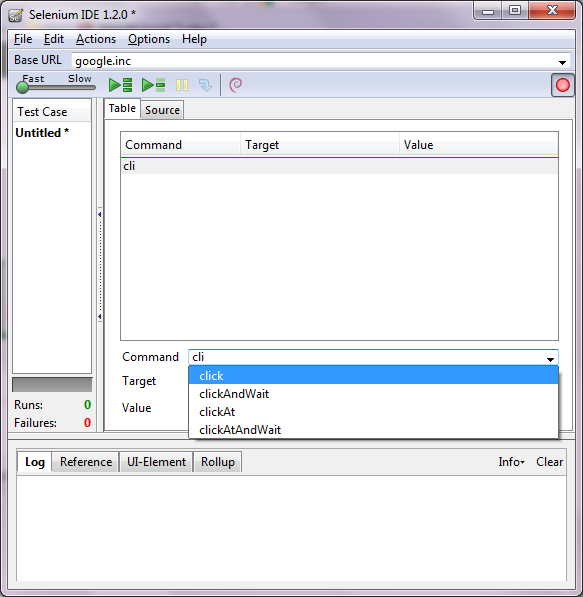


Figure 5.8 : Command writing and auto completion

Other features of Selenium include

* Capacity to save tests in Selenese, ruby scripts along with other languages
* Intelligent field selection
* Walk through tests
* Selenium user-extensions file support

### Cost & Complexity

Selenium is an open source tool thus the infrastructure cost of setting the software is nil. In order to work on selenium IDE , the user needs to know HTML . Thus, the complexity of the testing the system using Selenium is less.

### JUnit (www.junit.org)

JUnit is an open-source unit testing framework written in Java, for testing applications written in Java programming language. It is a simple and straightforward tool that helps developers to unit test their components and modules during the development phase .The test methods are identified using annotations. JUnit tests can be run using Eclipse or other equivalent IDE by including the JUnit jar file.

### Features –

1. Downloadable as a zip or jar file from [www.junit.org](http://www.junit.org). Tests can be run using Eclipse or equivalent IDE by including the JUnit jar file in the build path.

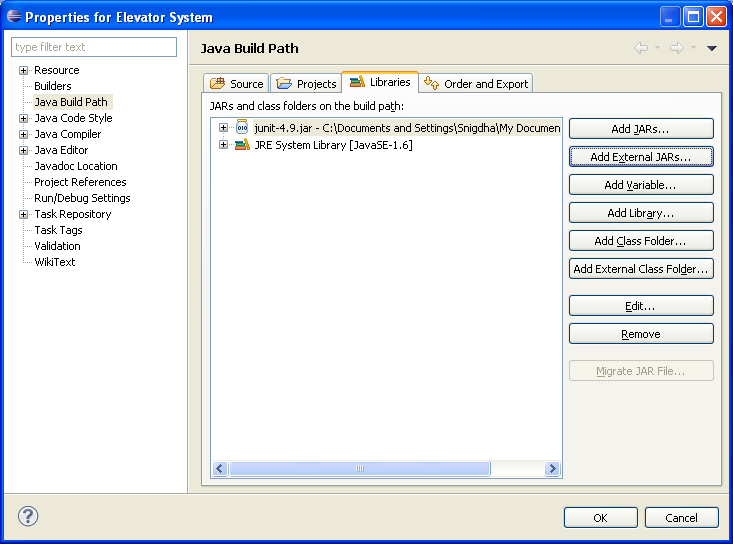


Figure 5.9 : JUnit Build path modification

1. Facilitates writing and running of repeatable tests. Allows testing of the components and modules while developing. Developers can implement a feature and test the same before proceeding to the next feature.

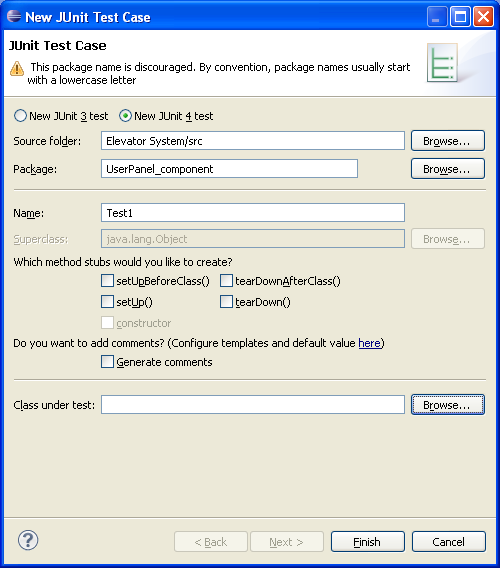


Figure 5.10 : New JUnit Test Case

1. Straightforward – methods can be designated as test methods using annotations. Some of the important Annotations are –

|  |  |
| --- | --- |
| **Annotations** | **Description** |
| **@Test** | Identifies that the method is a test method |
| **@Before** | Specifies that the method associated with the @Before annotation would be run before each test. Used for preparing the test environment. |
| **@After** | Specifies that the method associated with the @After annotation would be run before each test. |

Table 5.1 : JUnit Annotations

1. Expected result can be compared with the actual result, and the test can be marked as a Pass or a Fail accordingly using Assert methods. Some of the important Assert methods are –

|  |  |
| --- | --- |
| **Assert Method** | **Description** |
| **assertEquals(message, expected, actual)** | Asserts that the Expected and actual value are equal. |
| **fail(message)** | Forces failure with or without message. |
| **assertTrue(message, condition)** | Asserts that the condition is true. |
| **assert False(message, condition)** | Asserts that the condition is false |
| **assertNull(message, object)** | Asserts that the object is null. |
| **assertNotNull(message, object)** | Asserts that the object is not null. |

**Table 5.2 JUnit Assert Methods**

1. Multiple tests can be run concurrently.
2. Separate tests are independent of the others.

### Cost & Complexity –

It is available as an Open Source and there is no cost associated with creating the tests. Open Source IDEs may be used to run the JUnit tests, and so there is no cost associated with running the tests either. JUnit is easy-to-use and has a user friendly GUI. It is straightforward to understand, and users do not have to spend a lot of time learning the tool. Complexity of the JUnit tool is low.

### ElementTool

Elementool was founded in 2000. Today, the company has risen in the ranks to become one of the leaders in SaaS solutions all over the world. The Elementool is an online testing management tool. The development team and the testing team both use the Elementool during the project and use this as a medium to communicate among themselves. The test cases can be created in the Elementool and once they are tested, if any bugs are found they can be raised with the “Issue” option. These raised bugs can then be notified to the concerned developer, the status of the test case result/ bug keeps changing until it is completely fixed. Elementool will keep logs of every report and documents that has been send between teams. The Elementool can be used by the managers of the project to track the progress of the project, by which an estimate can be made on the completion time of the project and they can also track the resources, if any extra resources are needed they can be added to the project or removed from the project if not required.

### Features:

* From the initial stages of the project development process, Elementool enables software testers to track test cases and software defects.
* The Elementool bug-tracking tool helps you to track and save the discovered bugs and defects.
* The customer calls are directly managed through help desk management service. These calls are tracked through the user account in the Elementool website.
* Using Elementool web site, you can reply and manage the support calls submitted by your clients.

**Getting started with Elementool:**

Elementool (URL: [http://www.elementool.com](http://www.elementool.com/)) is the provider of web based management tools.



**Register to Elementool**

A test project manager, or test lead is the candidacy to register to Elementool. He later adds several users to the project.

The below picture shows the main page of Elementool with the services provided to the users.



Figure 5.11 Elementool Main Page

Once a user logins to elementool, user is shown with updates relating to any test cases or issues assigned to him. Any issues raised by the user are shown and other basic options.

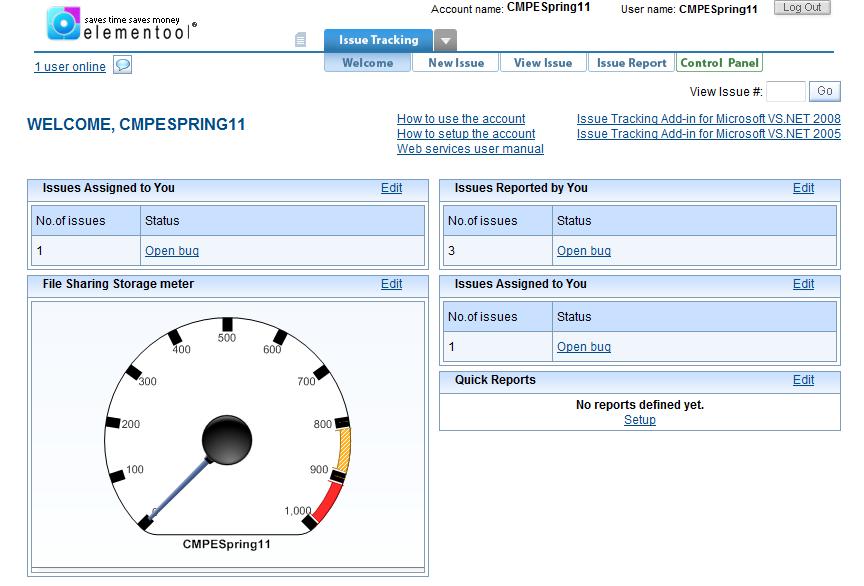


Fig 5.12 Elementool User Home Page

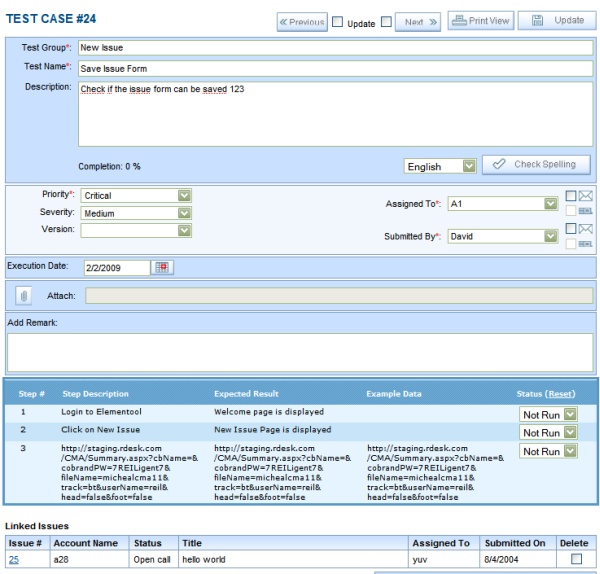
**Test Cases:**

The Test Cases option allows users to:

* Create New Tests – Allows creating new tests and assigning them to the project team i.e. developers, test engineers etc.
* Edit Existing Tests – Allows editing the tests.
* View Tests – Allows viewing all the test cases.
* Test report – Test case report shows the summary of the run of each test case
* Control Panel – Allows editing test case form, user profiles and user accounts.

Other uses like copying and moving test cases are also provided

The below picture shows the test case template in Elementool:



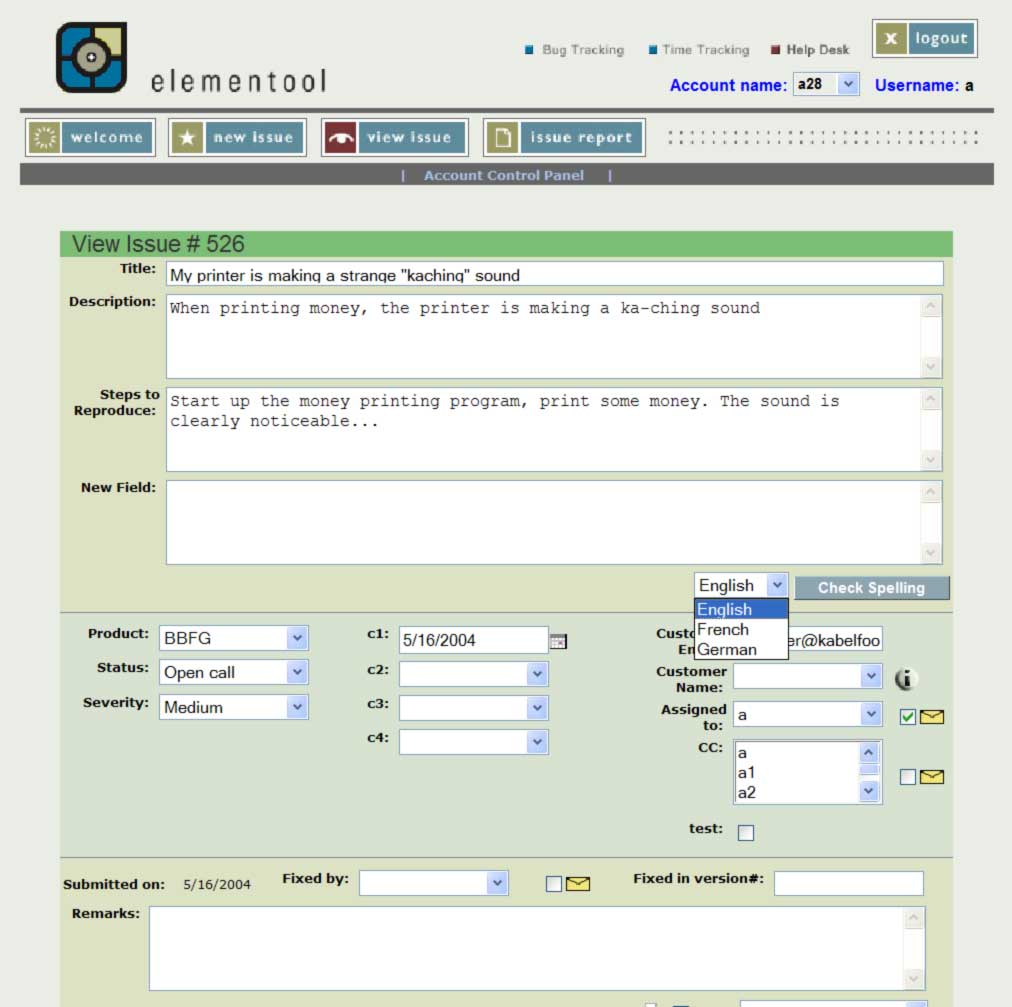
**Fig 5.13 Elementool Test case Template**

Once a test case has been written in the Elementool, the Lead assigns it to the Test Engineer, he executes the test case and if he finds a bug in any test case he then raises an issue. The issue raised can be assigned to the Lead who then assigns it back to the developer who is concerned. The developer can then look into the issue and perform the necessary operations and update it back to the lead. During the process everyone is updated with any changes made to the test case or issue.

**Issue Tracking:**

It contains the below options:

* New Issue: Allows users to create new Issue for a certain test case.
* View Issue: Allows viewing a particular issue raised.
* Issue Report: Shows the complete report of each issue raised.
* Control Panel: Allows to edit issue form, edit accounts, edit user profiles



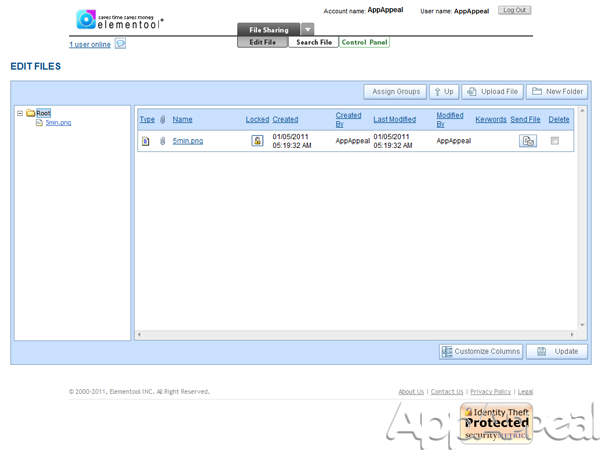
**Fig 5.14 Elementool “View Issue” screen**

**File Sharing:**

Elementool allows users to share files among them. It contains the following options:

* Edit File: Allows user to edit a particular file.
* Search File: Allows user for searching the file among all the files
* Control Pane: Allows users to edit user profiles and edit accounts.

The below picture shows the File sharing page.



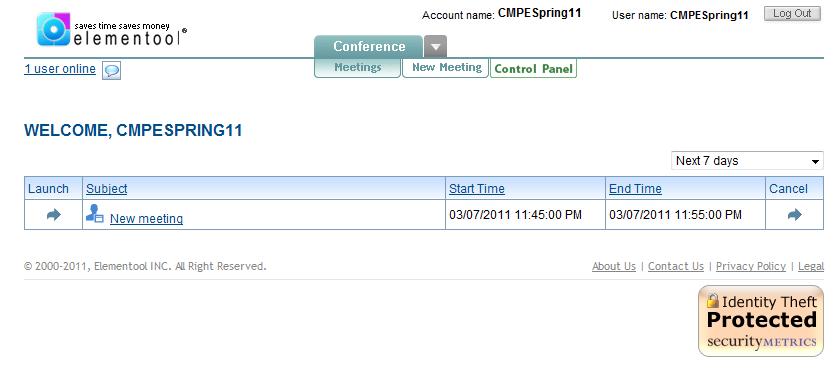
**Fig 5.15 Elementool “File Sharing” screen**

**Conference:**

The Elementool allows the user to create an online meeting which needs to make an appointment in the Meeting tab. The other user can come to see when the meeting begins, so they can join the meeting in time.

The conference option contains:

* Meetings – Displays the list of all meetings
* New Meetings – A new meeting can be made
* Control Panel – Allows to edit accounts, manage hosts and edit user profiles

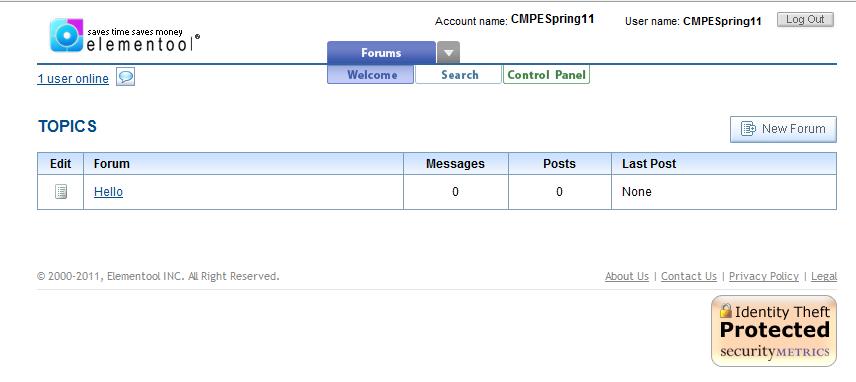


**Fig 5.16 Elementool “Conference” Option screen**

**Forums:**

The users in Elementool are provided with a feature to create Forums in the tool through which discussions can be performed among team members.

The picture below shows the Forums page:



**Fig 5.17 Elementool “Forums” screen**

Other [Elementool](http://www.appappeal.com/app/elementool/) Features like Docs and Help Desk exist. The help desk is used to edit user accounts, edit user profiles and many other options.

Elementool comes up with many advanced options such as:

* Mass update
* Time Tracking
* Sms setting
* Download database backup and etc.

### Cost and Complexity:

Elementool is not an open source tool but it does come with a Free Trial facility that lasts for 60 days. The cost of Elementool can get a little high, which could deter smaller businesses from trying it out. The elementool can be purchased based on the facilities that the user wants, each facility can be separately purchased. For example, the time tracking software costs about $90 per month for one project or $30 per project per month with a minimum of $120 per month total for the per project fee. Issue tracking service costs about $90 per month for a single account or $120 per month for two to four accounts. Every additional account costs another $30 per month. Elementool offers many other services, so it is easy to see how the user can begin to spend a great deal of money for services.

The elementool is a very user friendly tool and is very easy to use. Even though it has some advanced features the overall tool is kept very simple to make it very user friendlier and easily understandable. The complexity of this tool is low.

## TEST STANDARDS

The following is a list of standards that should be followed for performing system tests:

* Test design document should be prepared to capture the various test cases.
* There should be a test case associated with each requirement.
* The Test Design document should have a requirement traceability matrix so that each of the test cases can be traced back to the requirement it validates.
* Both black box and white box testing should be performed to ensure functional as well as logical correctness of the system.
* The test cases should be well documented according to the agreed format to make it easy to understand and reuse in future.
* Test data set should be selected so that maximum defects are captured.
* The test cases should be simple and independent of each other.
* Test cases from the previous version of the system should be reviewed in order to identify the test cases which could be reused once the current version changes are done.

## SUMMARY

Regression testing is mainly done to uncover new errors after a code change has been done to fix a bug. In this project, the plan is to add a new alarm component, modify the existing system by changing the algorithm, and modifying other components in the system like floor panel, user panel etc. and to perform the testing at the system, component and configuration levels. At the system level, the entire functionality of the system is checked to ensure that it meets specifications. Component level testing entails measuring the quality of the components that make up the system. Configuration testing checks the system behavior/response to change in the configuration/landscape. Testing at these different levels will help to guarantee a stable and robust system.

## Bibliography

*Elementool*. (n.d.). Retrieved 09 18, 2011, from Elementool.com: http://www.elementool.com/

Jerry Zeyu Gao, H.-S. J. In *Testing and Quality Based Assurance for component based Software.* Artech House.

*Selenium*. (n.d.). Retrieved 09 2011, from wikipedia.org: http://en.wikipedia.org/wiki/Selenium\_(software)

*Selenium IDE*. (2011, 09 14). Retrieved 09 18, 2011, from http://seleniumhq.org: http://seleniumhq.org/docs/02\_selenium\_ide.html

Junit. Retrieved 09 18, 2011,from http://www.vogella.de/articles/JUnit/article.html

Junit. Retrieved 09 18, 2011,from www.junit.org

<http://www2.sas.com/proceedings/sugi30/141-30.pdf>,Software Testing Fundamentals—Concepts, Roles, and Terminology byJohn E. Bentley, Wachovia Bank, Charlotte NC

Testing and Quality Assurance for Compnent Based Software- Jerry Zeyu Gao,Jacob,Tsao,Ye Wu,2003 Artech House Inc.

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)

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

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