CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

ADLER TEST MANAGER

A graduate project submitted in partial fulfillment of the requirements
For the degree of Master of Science in Software Engineering

By

Veronika Movagharian Pour

December 2015
The graduate project of Veronika Movagharian Pour is approved:

__________________________________________  _________________________
Son Pham, Ph.D.                                Date

__________________________________________  _________________________
Jeffrey Wiegley, Ph.D.                         Date

__________________________________________  _________________________
Adam B. Kaplan, Ph.D., Chair                  Date

California State University, Northridge
Dedication

I dedicate my thesis work to all my family members and friends whose help and endless support has been a vital encouragement for me throughout this path. I greatly offer my gratitude to my wonderful parents, Peter and Kathrin whose words of inspiration helped me discover my tenacity, perseverance and gave me the inner strength to tackle the rough patches this journey offered.

I also dedicate and am greatly thankful to my co-workers and my professors who helped a great deal in expanding my knowledge and technological skills required for achieving my goals in my career path.
Acknowledgements

I express my heartfelt gratefulness to Professor Adam Kaplan and Professor Robert Lingard who were generous with their time and expertise to help me throughout my work. A sincere thanks to Professor Adam Kaplan, my committee chairman for his countless hours of dedication and patience in helping me out through the entire work. Without their overwhelming help and support, it would have been impossible for me to complete this project.

I would also like to thank and acknowledge my thesis committee members, Professor Jeff Wiegley and Professor Son Pham, for demonstrating sincere interest and their contributions to this work. Over the past years each of them has given me that much needed invaluable guidance and insightful suggestions that has led me to this position where I am right now. I am fortunate to have such a talented group of outstanding professors.

Lastly, I would like to express my heartiest gratitude to my family and friends, especially my parents who have always been there with their constant love and much needed emotional support.
# Table of Contents

Signature Page ............................................. ii  
Dedication ................................................. iii  
Acknowledgements ........................................ iv  
List of Figures .............................................. vii  
Abstract .................................................. xi  

## Chapter 1 Introduction .................................................. 1  
1.1 Adler Test Manager Project Scope .................................. 1  
1.1.1 Adler Test Manager Users ........................................ 2  
1.1.2 Copies of Software under Test ................................... 3  
1.1.3 Adler Test Manager’s other Objects .............................. 3  
1.2 A Typical Scenario .................................................. 5  
1.3 Project Summary .................................................. 6  

## Chapter 2 System Architecture and Design ......................... 7  
2.1 System Architecture ............................................... 7  
2.2 System Design ................................................... 8  
2.3 System Functionality ............................................... 9  

## Chapter 3 Test Management Main Features ......................... 11  
3.1 Test Case Management Features .................................... 11  
3.1.1 Test Case Generation .......................................... 11  
3.1.2 Test Case Edit ................................................ 13  
3.2 Test Plan Management Features ................................... 13  
3.2.1 Test Plan Generation .......................................... 14  
3.2.2 Test Plan Edit ................................................ 15  
3.2.3 Test Plan Status Check ........................................ 15  
3.3 Test Execution Management Features ............................ 16  
3.3.1 Testing Test Case Step Status .................................. 17  
3.3.2 Testing Test Case Status ....................................... 18  
3.4 Bug Management Features ......................................... 18  
3.4.1 Bug Generation ................................................ 18
3.4.2 Bug Edit
3.4.3 Assigning Bug to Developer
3.4.4 Assigning Bug to Tester
3.4.5 Changing Bug State to Complete
Chapter 4 Version-Control /Code-Patching Features
4.1 Software Versions
4.1.1 Dev Software Copy
4.1.2 Test Software Copy
4.1.3 Release Software Copy
4.2 Code Change Announcement
4.3 Code Patching Management
4.3.1 Code Change Release Feature
4.3.1 Code Compare Feature
Chapter 5 Project Management Support Features
5.1 Task Management Features
5.2 Testing/Debugging Cost and Effort Estimation Support Features
Chapter 6 Available Features for Each User Type
6.1 Available Features for Administrator
6.2 Available Features for QA Director
6.2.1 Test Case Management
6.2.2 Test Plan Management
6.2.3 Bug Management
6.2.4 Code Patching Management
6.2.5 Testing/Debugging Cost & Effort Estimation
6.2.6 Create New User
6.2.7 Change Password
6.3 Available Features for Tester
6.3.1 ToDo List Management
6.3.2 Bug Management
6.3.3 Password Management
6.4 Available Features for Developer
6.4.1 ToDo List Management
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.2 Bug Management</td>
<td>46</td>
</tr>
<tr>
<td>6.4.3 Code Management</td>
<td>46</td>
</tr>
<tr>
<td>Chapter 7 Adler Test Manager Unique Characteristics</td>
<td>49</td>
</tr>
<tr>
<td>7.1 Test Management Unique Characteristics</td>
<td>49</td>
</tr>
<tr>
<td>7.2 Version Control Unique Characteristics</td>
<td>49</td>
</tr>
<tr>
<td>7.3 Project Management Support Unique Characteristics</td>
<td>50</td>
</tr>
<tr>
<td>Chapter 8 Conclusion and Future Work</td>
<td>51</td>
</tr>
<tr>
<td>8.1 Future Work</td>
<td>51</td>
</tr>
<tr>
<td>References</td>
<td>52</td>
</tr>
<tr>
<td>Appendix A: Database schema</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1.1 Adler Test Manager Use Case Diagram ......................................................... 2
Figure 1.2 Test Case State Diagram ............................................................................ 4
Figure 1.3 Bug State Diagram .................................................................................... 4
Figure 1.4 Adler Test Manager Typical Activity Sequence Diagram ......................... 6
Figure 2.1 Software System Architecture .................................................................. 8
Figure 2.2 System Functionality ................................................................................ 9
Figure 3.1 New Test Case Generation Page ................................................................. 12
Figure 3.2 Test Case Edit Page .................................................................................. 13
Figure 3.3 Create New Test Plan Page ....................................................................... 14
Figure 3.4 Test Plan Edit Page ................................................................................... 15
Figure 3.5 Test Plan Status View Page ...................................................................... 16
Figure 3.6 Testing Test Case Page .............................................................................. 17
Figure 3.7 Testing Test Case Ready to Create a New Bug Page ................................. 17
Figure 3.8 Create New Bug Page ................................................................................ 19
Figure 3.9 Bug Edit Page ........................................................................................... 19
Figure 4.1 Code Change Announcement Page ............................................................. 23
Figure 4.2 Code Change Announcement Page (2) ....................................................... 23
Figure 4.3 Code Patching Management Page .............................................................. 24
Figure 4.4 Bug Edit Page Displayed by Clicking “Review Bug” Button ..................... 25
Figure 4.5 Code Compare Page When Comparing to the Current Version ............... 26
Figure 4.6 Code Compare Page with the Date Drop Down List .................................. 26
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 4.7</td>
<td>Code Compare Page When Comparing to an Old Version</td>
<td>27</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Create New Test Plan Page</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Bug Edit Page</td>
<td>29</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Administrator Control Panel Page</td>
<td>31</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Template Interface Page with List View</td>
<td>31</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Template Create New Page</td>
<td>32</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Template Interface Page with Table View</td>
<td>33</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>QA Director Control Panel Page</td>
<td>33</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Test Case Management Page with List View</td>
<td>34</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Test Case Management Page with Table View</td>
<td>35</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>Test Plan Management Page with List View</td>
<td>35</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>Test Plan Management Page with Table View</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>Bug Management Page with List View</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>Bug Management Page with Table View</td>
<td>37</td>
</tr>
<tr>
<td>Figure 6.12</td>
<td>Code Patching Management Page</td>
<td>38</td>
</tr>
<tr>
<td>Figure 6.13</td>
<td>Historical Data Display Page</td>
<td>39</td>
</tr>
<tr>
<td>Figure 6.14</td>
<td>Create New User Page</td>
<td>39</td>
</tr>
<tr>
<td>Figure 6.15</td>
<td>Change Password Page</td>
<td>40</td>
</tr>
<tr>
<td>Figure 6.16</td>
<td>Tester Control Panel age</td>
<td>40</td>
</tr>
<tr>
<td>Figure 6.17</td>
<td>Tester ToDo List Page with Table View</td>
<td>41</td>
</tr>
<tr>
<td>Figure 6.18</td>
<td>Tester ToDo List Page with List View</td>
<td>42</td>
</tr>
<tr>
<td>Figure 6.19</td>
<td>View Test Case Detail Page</td>
<td>42</td>
</tr>
<tr>
<td>Figure 6.20</td>
<td>Bug Management Page with Table View</td>
<td>43</td>
</tr>
</tbody>
</table>
Figure 6.21 Bug Management Page with List View ...............................................................44
Figure 6.22 Developer Control Panel Page ...........................................................................44
Figure 6.23 Developer ToDo List Page with Table View .....................................................45
Figure 6.24 Developer ToDo List Page with List View ........................................................46
Figure 6.25 Code Change Announcement Page ....................................................................47
Figure 6.26 Code Change Announcement Page (2) ..............................................................47
ABSTRACT

ADLER TEST MANAGER

By

Veronika Movagharian Pour

Master of Science in Software Engineering

Due to the wide availability of software products and the high compatibility among software companies in the business word, it is extremely important to maintain the high quality of software applications. Testing software products is the best way to ensure the conformity of software actual functionality and performance with the customer expected ones as it can reveal large number of Bugs introduced to the software system removing which will improve the underlying software quality attaining higher customer satisfaction.

Adler Test Manager is a test management tool that supports software testing and quality assurance. This role-based manual testing tool is a web-based application that provides the users with software testing features based on the user role offering services to four user-types: Administrator, QA Director, Tester and Developer allowing Test Cases, Test Plans, and Bugs generation, edit and usage to perform software testing activities.

Furthermore, Adler Test Manager has a built-in version control mechanism that uses relational database tables to keep track of code changes providing the ability to revert the code back to any previous code version. This mechanism allows Developers to announce code changes and the QA Director to patch changes. It also keeps track of Bugs allowing the association of the changes to the related Bugs.

Moreover, this manual test management tool supports project management by maintaining a Bug history mechanism assisting the project managers in estimating testing/debugging costs and efforts by allowing them to access and compare the estimated and actual costs and efforts spent on similar Bugs introduced to similar software projects. Using this mechanism enables the project managers to have closer estimations to the actual testing/debugging costs and efforts allowing better testing/developing tasks scheduling, planning and resource allocation.
Chapter 1

Introduction

Adler Test Manager is a web-based easy-to-use software management tool that helps software developers in the process of testing a software product and finding and fixing its Bugs. Adler Test Manager is a manual testing tool that allows the creation of Test Cases and Test Plans; keeps track of test results; assists in creating, managing and tracking Bugs. This tool also manages code patching and tracks changes to source code. Unlike other source control and Bug-tracking tools such as SVN/GIT and Microsoft Test Manager that control and track changes in files, Adler Test Manager tracks source code revisions using a relational database.

Moreover, Adler Test Manager is a role-based test management tool. Whereas other test management tools such as Microsoft Test Manager allow all users to access all pages, albeit-with some features disabled for unauthorized users, Adler Test Manager has a specific view designed for each user-type based on that user’s role. Thus Adler Test Manager displays only the features that each user has permission to access.

Furthermore, this test management tool assists project managers in estimating testing and debugging efforts and costs by allowing the creation of historical data that can be used for estimation purpose. Capturing detailed information regarding raised Bugs and the associated Bug-fixes, including the time spent on catching and removing Bugs provides a rich record of historical Bug data. This data can be used to estimate cost and effort of developing and debugging similar software products, or products with similar features.

1.1 Adler Test Manager Project Scope

Adler Test Manager provides its services to users with four different roles, including: Admin, QA Director, Tester, and Developer. To achieve high quality and integrity of the software under test, three copies of the software under test: Release Software, Test Software and Dev Software are being used during the software development life cycle. Test Case, Test Plan, and Bug are other objects that also exist in this system. Figure 1.1 displays the scope of the Adler Test Manager tool system.
1.1. Adler Test Manager Users

As mentioned before, four types of users communicate with Adler Test Manager tool: Admin, QA Director, Tester, and Developer. There is a set of features designed for each user-type. Upon sign in, the Test Manager not only authenticates the user, it also determines user-type. It then navigates the user to a web page appropriate to the corresponding user-type. The following is a short description of the activities each user-type is empowered to perform:

1. Admin: This is the only user-type who can access the source code of Adler Test Manager tool and modify the code.

2. QA Director: This is the user who can create, activate, and inactivate Test Cases (TCs), and assign/reassign them to Testers through Test Plans. QA Director is also able to assign Bugs to Developers to fix by determining the name of the Developer who should work on that specific Bug and later approve/disapprove the fixes (changes), patch the approved changes to the Release Software and set Bugs as complete. Or, he can instead, set Bugs to Defer state in order to be fixed in a later time. This user-type also has the ability to estimate testing and debugging efforts and costs by accessing the Bug repository of previous similar software products and similar features.

3. Tester: A user of this type receives TCs assigned to that specific user, tests the Test Software following TC Steps for each Test Case, passes/fails each TC Step and consequently, passes/fails the associated TC. A Tester creates (proposes) Bugs, where each Bug corresponds to a single failed Test Step. The tester then retests Test Software and verifies Bugs after appropriate fixes are applied.
4. Developer: Developers are assigned Bugs that they are tasked with fixing. After fixing each assigned Bug, the Developer records the names of the table and the key values of the piece of code that are changed, puts a short comment on what the changes are and why they were made, and then submits the Bugs as Ready To Test.

1.1.2 Copies of Software under Test

In order to maintain the quality and integrity of the software under test, we provide three copies of the software under test: Release Software, Test Software and Dev Software. Release Software is the original software in use. The Test Software is a copy of the Release Software that is used for testing. The Dev Software is a copy of the Release Software at development time. When an enhancement to the Release Software is proposed or a Bug is raised that needs fixes, the patches are not applied to the Release Software directly. Instead, Dev Software is the software that undergoes development. When enhancements or fixes to the Dev Software are proposed and reviewed, the patches are applied to the Test Software. Test Software then are tested to ensure the effectiveness and appropriateness of the patches. The original Release Software receives the fixed code (patches) only after successful retest and associate approval.

1.1.3 Adler Test Manager's other Objects

Beside the four user-types and the three software copies that communicate with Adler Test Manager tool, Test Case, Test Plan, and Bug are objects that also exist in this system. The following is a short description of each object:

1. Test Case: a Test Case is an order set of actions and expected results that are being used to test software functionalities. A Test Case can be either in active or inactive state.
2. Test Plan: a Test Plan is a set of active Test Cases assigned to Testers to be used for testing software under test. A Test Plan can be created anytime big changes are applied to the Dev Software under test to evaluate the software quality and performance.
3. Bug: A Bug is created when a Test Case fails or the software under test crashes. It is a description of the existing problem identifying the related failed Test Case if any.

As mentioned before, a Test Case can be either active or inactive by itself. However, it can be in many different states in each Test Plan, one at a time. Figure 1.2 displays all possible states a Test Case can be in. Note that only active Test Cases appear in a Test Plan.
Likewise, a Bug can also be in different states. The initial state, “Proposed” state, occurs when a Test Case fails and a corresponding Bug is generated. Figure 1.3 displays the states a Bug can be.
1.2 A Typical Scenario

The sequence of actions in a typical scenario that occurs in Adler Test Manager tool system triggered by the users, is as follows:

The QA Director creates TCs and activates the ones that are ready to use. Later on, this user creates a Test Plan which includes assigning some or all active TCs to Testers to be used to test the Test Software. (Only active TCs can be used for testing.) The QA Director can inactivate TCs whenever needed. This means if the QA Director inactivates a TC already assigned to a Tester, the TC will not be available to the Tester to use anymore.

When a Tester signs into the Test Manager, this user accesses the list of TCs assigned to him, opens one TC at a time and starts testing the Test Software by following the specified Steps in order. The tester passes or fails each Step based on the observed software behavior. The observed behavior is compared to the expected behavior mentioned in each Step and the Step is passed if the behaviors match. If a Step is failed, the Tester creates/proposes a Bug. A Bug contains a short explanation of the problem behavior and has a reference to the TC from which it is created.

The QA Director receives the submitted/proposed Bugs and assigns the Bugs to Developers to fix. However, the QA Director may also decide to postpone fixing some Bugs to a later time. These Bugs are Deferred until the QA Director assigns them to a Developer. To fix each assigned Bug, the Developer applies changes to some files in the Dev Software and lists the name of the changed files in the Bug document along with a short note explaining what changes have been made to each of those files and why. The Developer then submits the Bug as Ready For Test.

The QA Director receives the set of all fixed Bugs, reviews the fixes, applies the appropriate fixes to the Test Software and assigns associated Bugs to the Testers to test and make sure that the problem is fixed. If the problem is fixed, the Tester verifies the Bug. (If not, the Tester proposes the Bug which then, will be reassigned to a Developer to rework.) Figure 1.4 displays the sequence of the activities performed in a typical cycle.

Verified Bugs are received by the QA Director. During each Release, the QA Director patches all changed code related to the verified Bugs preserving the old content of the changed code in the backup table in case for any reason, there is a need to reverse to or access the old code.
1.3 Project Summary

This thesis documents the creation of Adler Test Manager, a useful tool that provides facilities to find and fix software bugs and to manage and track software changes to achieve high software product quality and estimate testing and developing efforts and costs for project management purposes. It has four user-types including Admin, QA Director, Tester, and Developer and uses three copies of the software under test; Release Software, Test Software, and Dev Software. Approved patches of each Test Plan are applied to the Release Software during that Plan release. Test Cases, Test Plans, and Bugs are created and processed through the users using Test Manager.
Chapter 2
System Architecture and Design

Adler Test Manager is a web-based application meaning that authorized users can access and use it through their local PCs or Laptops anytime from anywhere using internet connection via standard browsers such as Internet Explorer, Firefox, and Google Chrome. This web application can be accessed and used by multiple users simultaneously without any collusion.

2.1 System Architecture

Adler Test Manager is implemented based on a three-tier software system architecture: Client Tier, Application Tier, and Data Tier. And to develop the system, I have used LAMP Stack: Linux as the Application Server’s operating system, Apache Server for establishing required network communications, PHP as the server-side scripting web development programming language, and MySQL DataBase Management System (DBMS) as the Database Server system to manage data storage and retrieval. Figure 2.1 displays the underlying software system architecture.

As Figure 2.1 implies, in this system, the Client or User connects to the web Application (Adler Test Manager) located on the Application Server via a standard Browser using his personal PC or Laptop. The Apache Web Server takes care of establishing internet connection between the user’s PC/Laptop and the Application server. Once the connection is established, the user can access and use the Application through the represented Graphical User Interface (GUI) after successful authentication. The Application Server communicates with the MySQL DBMS in order to process the user request and represent the proper GUI to the user. The Database Server has all the code for displaying GUI web pages stored and by receiving the value of key parameters triggered by each user-side event (request), it fetches the proper web page code and returns it to the Application server to display the proper GUI to the user. The Apache Web Server also establishes internet connection between the Application server and the MySQL DBMS.
2.2 System Design

The Adler Test Manager Application which is located on the Application Server is only one .php file named AdlerTestManager.php that runs the whole system. However, in order to perform the system operation, the Application uses two database tables: Class table and GUI table.

Having used the template class in the class table, the AdlerTestManager.php evaluates the key values triggered by each request in each step, fetches the proper source code from the GUI table with the help of Database Server, and displays the associated web page (GUI) to the user by running the retrieved source code written in PHP. On the other word, the functionality of the entire system is based on interaction between the AdlerTestManager.php file, the Class table and the GUI table.
2.3 System Functionality

Figure 2.2 displays the system functionality. The interaction between the user and the Adler Test Manager starts when the user browses the associated link using a standard browser on his personal PC/Laptop leading the AdlerTestManager.php to run. Once this event takes place, the Application Server requests required classes from the Database Server. The Database Server retrieves and returns the requested classes. Once the classes are retrieved, the Login page, which is the default web page, is displayed to the user to log in. This is because the key parameters are set to the Login page values as default.

Figure 2.2 System Functionality

In the next step, the user enters the personal Login and Password and clicks the “Log In” button.
The Application Server authenticates the user using the User class and User table and if the process completes successfully, the proper web page with the available features to use based on the user role (type) will be displayed to the user. Each time the user clicks a button or a link, an event occurs by which the key parameters receive value and the system is triggered to fetch the proper code and to display the appropriate web page.

To implement this web application, I have used the package of Object Relational Database of Comp 440 (database design). This package includes the Template and Meta class tables, template class stored in the Meta class table, and a .php file that uses the template class and the Template table structure to fetch and display the GUI to user. The template class facilitates the Template table utilization.
Chapter 3

Test Management Main Features

The main purpose of Adler Test Manager is to support Software Quality by providing Test Management Features. These features are used by users with QA Director, Tester and Developer roles to verify that the software under test meets (performs) the required functionality in the correct manner by allowing the authorized users use the available features.

As we discussed before, each user has access only to those features that is allowed to use meaning that if a feature is not available to a user, that user is not authorized to use that feature. This chapter provides explanation for each of these test management features determining who has the ability to use each feature.

3.1 Test Case Management Features

A Test Case is a set of paired Actions and Expected Results in a specific order called Steps that are being determined during the Test Case generation. Each Test Case is designed to test the functionality of a feature and the Steps are the Actions that should be performed in the specified order and in each Step, the result of each performed action should be compared to the Expected Result of that Step to ensure that the software under test reflects the expected behavior.

3.1.1 Test Case Generation

The first feature for Test Case management is the Test Case Generation Feature. A user with QA Director Role has access to this feature through “New” button available on the Test Case Management page. Figure 3.1 displays the web page for generating a new Test Case.

As shown in Figure 3.1, the Test Case Generation web page has two parts: in the top part, the information regarding the Test Case can be specified and in the bottom part, the Steps of the Test Case are generated.

The top part is pretty straightforward to use. Either there is a text box to type the related info in for each item or there is a drop down to select a value from.

The bottom part, Test Step Generation part, contains two text boxes: in the first text box the user specifies the Action that should be performed in that Step, and in the second text box the user specifies the Expected Result of that Action. The Expected Result might be blank if obvious. However, the Action text box cannot be blank, or the Test Step will not be created.
Once the Action is typed into the related text box and the Expected Result (if desired) is entered, clicking the “Add” button will add that Test Step at the end of last previous Step. If there is no previous Step, the new Step will be created as the first Step. The Steps are shown in the created order.

As mentioned before, each Test Case contains a set of Test Steps in a specific order. Each Test Step is generated by determining the Step Action and optional Expected Result one after the other until all Steps are created.

Each added Step can be modified or deleted after being added to the Step list using “Edit” and “Delete” button present in that Step next to the Expected Result column. However, remember that the Step Action cannot be blank or the changes will not be saved.

After we have determined the Test Case required fields with relevant data, typed or selected from the options provided, and the Steps are specified, the entire Test Case can be saved by clicking the “Save” button. If the “Cancel” button is clicked, the entire effort will be thrown away meaning that the new Test Case is not created and not saved in the database.

Once a Test Case is generated, it cannot be deleted. It is possible to edit an existing or pre-generated Test Case, but not to delete any saved Test Case. However, if a Test Case is determined as absolute, it can be inactivated. This can be done by editing the Test Case Status from “Active” to “Inactive”. This way, though the Test Case exists, it will not appear in the active and in-use Test Case list unless it is activated again. Any absolute Test Case might become useful in generating new Test Cases for the same or a similar project.
3.1.2 Test Case Edit

A generated and saved Test Case can be edited by selecting the desired Test Case name from the provided Test Case name list and hitting the “Edit” button next to the list. This event will direct the user to the Test Case Edit page with the text boxes’ and options’ values present there allowing the user to edit any of the desired fields. Each Test Step has separated “Edit” and “Delete” buttons that perform the related operation only on that Test Step. A new Test Step can also be added to the list of the existing Steps the same way as when generating a new Test Case. Figure 3.2 displays the web page for editing a new Test Case.

Once all desired changes are performed and the Test Case is ready, clicking “Save” button will cause the changes to be saved. Like the time when creating a new Test Case, the “Cancel” button will undo the changes performed so far. However, please note that once the changes are saved using the “Save” button, there is no way to return the Test Case to the previously existing Test Case unless by performing another edit and retying and/or reselecting (returning) the values to the previous values through the use of the Edit page and clicking the “Save” button to save the current changes.

![Test Case Edit Page](image)

Figure 3.2 Test Case Edit Page

3.2 Test Plan Management Features

A Test Plan is a set of active Test Cases that are determined to be tested in each period that is specific to its create date. The Test Plan create date is automatically set to the date when the Test Plan is created. It can then be used to access the desired Test Plan.

Test Plan is being created by a user with QA Director role. This user can also modify or check its Status (status of each Test Case of the Test Plan). This user type is the only user who has full access to the Test Plan with modification feature.
A Tester can only access to the Test Cases of each Test Plan that are assigned to this user for testing. A Tester then can open each assigned Test Case for each period and start with testing. This user can pass or fail a Test Case in any of its steps and in case of failing, the Bug generation feature gets activated allowing the Tester to generate a Bug. The Tester then passes or fails the whole Test Case.

A Test Plan life starts the time when a user with QA Director Role creates it and ends when the Test Plan status turns to complete. During a Test Plan life cycle, the QA Director can access the Test Plan and modify it. Upon saving the modification, the changes will be reflected. QA Director can also check the Test Plan Status by selecting any desired time regardless of whether the Test Plan is complete or not.

### 3.2.1 Test Plan Generation

As mentioned before, a Test Plan can be generated by QA Director. This user type has the feature designed to access the Test Plan Generation page. In the first step, the QA Director selects the project name for which the Test Plan is going to be generated. Then, in the next step, the button “Create Plan” is clicked. Upon clicking this button, the QA Director is directed to the Test Plan Generation page. Figure 3.3 displays the Create New Test Plan Page.

![Create New Test Plan Page](image)

**Figure 3.3 Create New Test Plan Page**

The Test Plan Generation page lists all the active Test Cases related to the selected project allowing the QA Director to select the ones that are needed to be tested for this period and to assign a Tester to Test. To select and assign a Test Case, the QA Director selects a Tester from the provided Tester drop down list and then, clicks the “YES” button next to that Test Case so that it's selected. Once the “YES” button of a Test Case in the list is selected, that Test Case is added to the Test Plan if a Tester is assigned to it. If the “NO” button is clicked, the associated Test Case is removed from the Test Plan.
The Test Plan is actually created once the save button is clicked. This is the time when the current date is assigned to the Test Plan and the Test Plan is officially created. Once a Test Plan is created, it cannot be deleted, but can only be inactivated meaning it is canceled. Moreover, if the Plan gets canceled before being saved, the Test Plan will not be created at all.

3.2.2 Test Plan Edit

A Test Plan can be modified only by a user with QA Director role. In order to modify a Test Plan, the QA Director needs to select the project name and then the create date from the provided drop down list on the main page and click the Edit button which will direct this user to the Test Plan Edit Page displaying the list of all active Test Cases associated with the selected project determining the selected Test Cases along with the specified Tester to perform the test. Figure 3.4 displays the Test Plan Edit Page.

![Figure 3.4 Test Plan Edit Page](image)

The QA Director can remove any selected Test Case form the Test Plan or select any not selected ones having determined a Tester for each. However, it is important to know that a Test Case that is already tested and has a pass/fail Status, cannot be removed from the Test Plan. Only Test Cases with null Status value are allowed to be removed from a Test Plan. Moreover, a Test Plan with complete Status cannot be updated. It can only be reviewed by the QA Director.

3.2.3 Test Plan Status Check

QA Director is also provided with the feature that allows this user to view the Status of a Test Plan any time. This feature allows the QA Director to see what the Status of each Test Case of the selected Test plan is; pass, fail, blank. Blank Status means that the Test
Case is not tested yet if that Test Case is included in the selected Test Plan. Figure 3.5 displays the Test Plan Status View page.

**Figure 3.5 Test Plan Status View Page**

In order to view a Test Plan, the QA Director selects the project name and the create date from the associated drop down list and clicks the View button. This will direct this user to the Test Plan Status View page where the list of the Test Plan Test Cases along with their Status is presented.

### 3.3 Test Execution Management Features

Once a Test Plan is created and Test Cases that need to be tested in the Test Plan are determined and assigned to Testers, each Tester can then access and perform only the Test Cases assigned to that Tester. This means that upon successful login, each Tester can access Test Cases assigned to him/her via displaying links for ToDo List Management.

By clicking one of the ToDo List Management links, the Tester accesses the ToDo List Page in which the assigned Test Cases are listed ordered by their priority ready to test by clicking the associated button to access each Test Case.

When a Tester clicks a button associated with a Test Case to access the Test Case, the Testing Page will open up allowing the Tester to see the selected Test Case Steps containing Action and Expected result one by one along with two buttons: Pass and Fail on the top of the page. On the bottom part, the Login page of the Web Application (Software) that the test should be performed on will be displayed allowing the Tester to start with testing the Web Application following the selected Test Case Step. Figure 3.6 displays the Testing Test Case page.
Figure 3.6 Testing Test Case Page

3.3.1 Testing Test Case Step Status

Once accessing Testing Page, the Tester needs to perform the Action specified in each step in order and determine the Test Step status by comparing the Expected Result of that Step to the actual Test Result viewed. If they match, the Tester passes the Test Case Step by clicking the Pass button. Otherwise, the Fail button should be clicked. By clicking the Fail button, Create Bug button along with a Note textbox shows up allowing the Tester to write a note and create a Bug regarding to that Test Case and the Failed Test Case Step. Figure 3.7 displays the Testing Test Case ready to create a new Bug page.

Figure 3.7 Testing Test Case Ready to Create a New Bug Page
The Create New Bug button will direct the Tester to the Bug Generation page in which the Test Case along with its Steps including the failed Step will be present and the Tester can explain the issue in the specified detail section further. The Note also will be present in the New Bug generation page. The Tester then submits the Bug by clicking the Save button.

3.3.2 Testing Test Case Status

If a Test Case Step fails, the entire Test Case Fails and upon saving the testing Test Case by the Tester, the Test Case Status will be Fail. Thus, once a Test Case Step is Fail and a Bug related to the existing issue is created and saved, the system fails the Test Case without continuing the rest of the Steps. In this case, any Test Case Steps that have no Status associated with, will have Status None. The failed Test Cases will be tested again once the associated Bug is fixed and the fix is approved.

On the other hand, if all Test Case Steps have Pass Status upon saving the Test Case, the Test Case Status will be determined as Pass. The passed Test Cases are complete and require no further action.

3.4 Bug Management Features

When a Test Case fails, “Create New Bug” button appears on the screen clicking which will direct the Tester to the Create New Bug Page. The Failed Test Case title will be displayed under “Test Case” and the Test Case Steps to regenerate the issue will be displayed under the “Bug Reproduction Steps” section.

A Bug can also be created without requiring to a Test Case Step to be failed. A User with Tester or Developer Role can access Create New Bug Page directly from the User Main Page. However, in this case, no Test Case title and Test Case Steps with the testing results will be associated with this Bug.

3.4.1 Bug Generation

Once in the Bug Generation Page, the Bug Title and the Root Cause of the Bug need to be specified in the provided Textboxes. In the Status section, “Assigned To” portion is specified as “Unassigned” in this page and the State is “Proposed” by default. In the Classification section, the Bug Area and Feature are specified. The Planning section of the new Bug can be specified by the QA Director later on though the Bug proposer, a user with Tester or Developer Role, can specify values for the planning section.

Once the general information about the new Bug is specified, the detail of the Bug can be added under the “Bug Detail” section in the provided Textarea. At this point, clicking the Save button will create the new Bug assigning it with a unique Id. The user will be specified as the Bug creator automatically without requiring any further action. Clicking
Cancel button will call off the new Bug generation. This button will also reset the associated Test Case test results if any. Figure 3.8 displays the Create New Bug Page.

![Figure 3.8 Create New Bug Page](image)

### 3.4.2 Bug Edit

Once a Bug is created, it can be accessed by the Tester or Developer to be modified. No matter who is the Bug author, any user with Tester or Developer Role is able to modify only the fields that are available to specify during the Bug generation by the user. However, the QA Director is able to modify all fields if necessary. The Bug Author is the only field that cannot be modified even by the QA Director. It is being set only during that Bug Generation automatically and can never be modified. Figure 3.9 displays the Bug Edit Page.

![Figure 3.9 Bug Edit Page](image)
3.4.3 Assigning Bug to Developer

Any newly created Bug is automatically set to the “Proposed” State. The QA Director is the user type who can assign the proposed Bugs to a Developer to fix. Once this change is saved, the Bug State will change from “Proposed” to “Dev”. Or, the Bug can be deferred by saving the Bug with the “Differ” State available to the QA Director. The cancel button however, will dismiss the change and the QA Director will be redirected to the previous page.

Upon assigning a Bug to a Developer to fix, the Bug will be displayed in that Developer’s ToDo List from which the Developer can access the Bug and start with the debugging activity. The Developer uses the Dev Software Version for debugging purpose. After debugging is successfully performed, the Developer specifies the name of the classes that are modified to fix the Bug along with some short explanation in the “Fix Detail” section. Upon saving, the Bug State changes from “Dev” to “Ready for Test” allowing the QA Director to patch the related changes to the Software Test version and then, assigning to a Tester to test whether the Bug is fixed.

3.4.4 Assigning Bug to Tester

Once a Bug is assigned to a Tester, the Tester will be displayed with the Bug and the related Test Case if any in the ToDo list. The tester then needs to find out which Test Case if any is associated with the Bug and the Test Case Steps leading to the underlying issue generation by reading the “Test Case” and the “Bug Reproduction Steps” sections. The “Bug Detail” section can also be read to gain more information about the Bug.

In the next step, the Tester can open the Test Case from the ToDo list and redo the Test following the Test Case Steps one by one like a regular Test Case. If the Test Case passes, the Bug is fixed and the Tester passes the Test Case by passing all the Test Case Steps, modifies the “Test Detail” section of the Bug to specifying that the Bug fix is verified, and changes the Bug State to “Verified”. Upon saving, the Bug state changes from “Test” to “Verified”.

3.4.5 Changing Bug State to Complete

A verified Bug can be set to the “Complete” State by the QA Director. In this case, the changes will need to be patched from the Software Dev Version to the Release Version. However, the QA Director may also reassign the Bug to a Developer if the fix is not acceptable regardless of whether the fix works or not if there’s a better solution to address the underlying issue. In this case, the Bug will be set to “Dev” State again and the specified Developer will see the Bug in his/her ToDo List.
Chapter 4

Version-Control / Code-Patching Features

Adler Test Manager has a built-in version control mechanism that allows tracking the code change providing the possibility of reverting changed code to any previous code version available in the archive. The built-in version control mechanism stores a copy of the currently running piece of code in the back-up table before applying the new/changed code to the release software version. This way, if by any chance the new code fails the regression tests or there is a need to go back to any of its previous versions for any possible reason, the desired version can be fetched from the back-up table and be replaced with the new/changed code while archiving the new/changed code in the back-up table.

Adler Test Manager Version Control starts with the code change announcement by Developer followed by patching the announced changes from Dev environment to Test environment and later releasing the changes to the Release Software. This section provides more details about the Adler Test Manager built-in Version Control Mechanism.

4.1 Software Versions

As discussed before, three copies of the software under test: Release Software, Test Software and Dev Software are being used during the software development life cycle. Version control mechanism targets all three software copies but in a deferent sense for each copy. In the following, version control features are further discussed for each software copy in a separate section.

4.1.1 Dev Software Copy

Any time the software under development requires development and a user with Developer role receives a development task, the developer performs the work on the Dev Software. Dev Software is patched with the successfully tested and approved new/changed code within a specific time period in a regular basis depending on the development speed and sprint duration, (the time period specified to develop some features and/or functionalities into the underlying Software), usually every two weeks. Therefore, it is extremely important for every developer to maintain a copy of under development code pieces on his/her local computer to avoid any code loss. However, it’s a good idea to announce code patching occurrence every time before happening just in case a developer forgets about the patching occurrence.

Any time a Developer creates new piece(s) of code or applies some changes in the existing code piece(s) regarding a Dev task, he/she uses the Adler Test Manager built-in version control mechanism to announce the name of the associated new/changed code
class and function pair(s) so that this code gets tested later on by the tester(s). The announced class and function pair(s) are ingested into the Test Software copy through using this built-in version control mechanism so that the code can be tested to ensure the functionality of the integrated code.

Moreover, the built-in version control mechanism is used by the QA Director to patch approved new/changed code into the Dev Software. This mechanism keeps the name of the new/changed class and functions pair(s) announced by developer(s) in the change_announcement database table allowing the QA director to push them into the Test Software copy for testing and to push them into the Release Software copy in a regular time period basis so that all software copies are up-to-date with the current approved changes and the Release copy after receiving those changes. Once the approved code is patched into the three software copies, it is removed for the change_announcement table.

4.1.2 Test Software Copy

Once some changes are performed in the Dev Software copy and the QA Director decides that it’s the time to apply the changes in the Release Software copy, he/she patches the changes announced in the change_announcement table to the Test Software copy for testing. In this stage of the Software life-cycle, if required Test Cases to test the changes already exist, the QA Director creates a Test Plan including all the related required Test Cases assigning them to Testers to test them. However, if some Test Cases that can test the functionalities and/or features related to the changes are missing, then, the QA Director creates new Test Cases and includes them in the new Test Plan. Thus, when a Tester is assigned Test Cases to use for testing the underlying Software, the code changes are already patched into the Test Software copy through the Alder Test Manager built-in Version Control system and the Test Software Copy is ready to use for testing.

4.1.3 Release Software Copy

When a Test Plan is complete and all associated Test Cases are passed or deferred to a later time, the QA Director patches the code of all the associated class and function pairs from the Dev Software copy to the Release Software copy and then, all the class and function pairs’ code from Release Software copy to Dev and Test Copies if required to ensure the consistency among the copies. The QA Director announces everyone the patching time before patching so that every Developer makes a copy of his/her work/code preventing any code loss and rework. The built-in version control system enables the QA Director to perform the code patching from Dev to Test and later on to Release copies, and also From Release copy to the Dev and Test Software Copies if needed.
4.2 Code Change Announcement

Once Developer finalizes the code changes to fix a Bug or implement an enhancement, the class and function pairs of the modified code need to be announced by the Developer. Figure 4.1 displays the Code Change Announcement page.

![Figure 4.1 Code Change Announcement Page](image)

As figure 4.1 implies, the first step in announcing code changes is selecting the Bug from the Bug drop down list followed by clicking the “Start” button. Upon clicking this button, the Project name associated with the selected Bug is specified and a drop down list representing the Project’s class and function pairs list existing in the Dev Software copy appears on the screen for the Developer to choose from. “Announce Change” button performs the announcement for the selected pair. Figure 4.2 displays the Code Change Announcement page when the class and function pairs drop down list is available to use.

![Figure 4.2 Code Change Announcement Page (2)](image)
Code Change Announcement also displays the list of all not released class and function pairs announced by the Developer. The “Delete” button associated with each announced class and function pair allows the Developer to call back that announced pair. The Developer can announce code changes for more than one Bug by selecting each Bug and announcing code changes for that Bug using “Announce Change” button followed by the next Bug code change announcement. The “Developer Control Panel Page“ link allows the Developer to go back to the Developer Control Panel page once done with the code change announcement.

4.3 Code Patching Management

Code Patching Management, a feature available to the QA Director, allows patching code from Dev Software Version to Test and Release Software Versions. There is also a feature that allows comparing the proposed changed/enhanced code by the Developer to the currently in use code version or to any old version stored in the backup table. Figure 4.3 displays the Code Patching Management page.

The Code Patching Management page displays the list of all not released announced class and function pairs along with the associated Bug and Project names in a table view. Column “Ready To Release” specifies which pairs are ready for patching. The QA Director can click on the “Ready” button associated with each class and function pair as a confirmation that the change is approved. Once this button is clicked, the “Ready To Release” column will have value 1 and the “Ready” button changes to “Not Ready” button clicking which will undo the approval action performed earlier. The “Review Bug” associated with each class and function pair allows the QA Director to view/edit the related Bug details in the Bug Edit Page. Figure 4.4 displays the Bug Edit Page available through clicking a “Review Bug” button. Once done with the Bug review, the “Code Patching Management Page” link will display the Code Patching Management page.
4.3.1 Code Change Release Feature

There are two buttons designed to perform the code change release operation: “Release To Test” and “Release” buttons clicking each performs the release to a Software Version.

“Release To Test” button performs patching the ready to release announced class and functions’ code from the Dev Software Version to the Test Software Version upon clicking. The completion of this operation is confirmed by changing the value of column “Released To Test” from 0 to 1 after which the QA Director assigns the associated Bugs to Testers to verify the effectiveness of the changes.

Once the changes associated with Bugs are verified by Testers, The QA Director can patch the changes to the Release Software Version by clicking the “Release” button on the Code Patching Management page. Once changes are released, the released class and function pairs will be in complete status and will be removed from the code change announcement table being displayed on the Code Patching Management page.

4.3.2 Code Compare Feature

There are two buttons along with a drop down list designed to allow compare the code. The drop down list contains the list of all announced class and function pairs with the name of the associated Project next to each pair allowing the QA Director to select one from the list followed by clicking one of the buttons displaying next to the drop down list; “Compare to the Current Version” and “Compare to an Old Version”.

Clicking “Compare to the Current Version” button will display the current code version of the selected class and function pair against the Dev code version of the same
pair next to each other allowing the QA Director to see the changes. Figure 4.5 displays the Code Compare page when comparing the changed code to the current code version.

![Figure 4.5 Code Compare Page When Comparing to the Current Version](image)

Clicking the “Get This Version” copies the Release Version of the selected class and function code to the Dev Software so that it can be tested after patching to the Test Software, and then, if it works, the current version of the class and function code in the Release Software can stay without change.

Clicking “Compare to an Old Version” button will display a list of create date of the selected class and function for the QA Director to select from. Figure 4.6 displays the Code Compare page with the date drop down list available to choose from.

![Figure 4.6 Code Compare Page with the Date Drop Down List](image)
Once the desired date is selected from the list and the “Compare” button is clicked, the code of the class and function created on the selected date will be displayed next to the code of the class and function of the Dev Software. Figure 4.7 displays the Code Compare page when comparing the changed code to an old code version.

Figure 4.7 Code Compare Page When Comparing to an Old Version

“Get This Version” button will overwrite the code in the Dev Version. This code can be tested once the Test Software code is also overwritten. “Code Patching Management Page” link will direct the QA Director to the page that the name implies.
Chapter 5

Project Management Support Features

Adler Test Manager is not just a testing management tool but also a project management support tool. It not only provides features for Test Case, Test plan and Bug management but also task management features allowing to assign/unassign Test Cases and Bugs to Testers for testing the under test software and to assign/unassign Bugs to Developers to fix software issues.

Moreover, Adler Test Manager has features to perform better testing and debugging cost and effort estimation to support project managers with task scheduling and resource allocation. The features enabling the storage and retrieval of historical data associated with the Bugs provides valuable information allowing closer estimations of required testing and debugging costs and efforts to the actual costs and efforts. This section provides more detail about Adler Test Manager’s available features for project management support.

5.1 Task Management Features

One task management feature embedded in the Adler Test Manager enables a user with QA Director role to assign Test Cases to Testers as testing tasks through Test Plan. Whenever creating a new Test Plan, the Tester drop down displays the list of Tester’s name to select from having the first name in the list as the default Tester. As visible in figure 5.1 which displays the Create New Test Plan page for a sample project, this drop down list is available for each Test Case in the Tester column. Upon saving a Test Plan, each testing task will be assigned to the specified Tester. The underlying task manager also allows assigning a testing task to a deferent Tester later in the Test Plan Edit page.

Figure 5.1 Create New Test Plan Page
The next task management feature enables the QA Director to assign a Bug to a user with Developer role to fix and later to a user with Tester role to test in the Bug Edit page. When a Bug is proposed by a Tester or Developer, the QA Director can assign the Bug to a Developer by selecting a Developer from the “Assigned_to” drop-down list in the Bug Edit page, selecting “Dev” State, then clicking the “Save” button. Upon saving, the Bug task will be appeared in the assigned Developer’s ToDo list. After fixing the Bug and proposing code changes by the Developer, the QA Director can assign the Bug to a Tester the same way using the “Assigned_to” drop-down list and selecting “Test” State. The Assigned_to and the State drop-down lists are visible in figure 5.2 which displays the Bug Edit Page.

![Bug Edit Page](image)

**Figure 5.2 Bug Edit Page**

### 5.2 Testing/Debugging Cost and Effort Estimation Support Features

Adler Test Manager allows the QA Director to store the estimated testing and debugging efforts per hour using the “Original Estimate” textboxes under Fix Effort and Test Effort sections. By considering the sum of the cost of assigned Developer and Tester per hour and multiplying each to the estimated required effort, we have the estimation of the required costs for debugging and testing the associated Bug.

Once a Bug is fixed and the code changes are announced, the Developer enters the actual spent effort in the “Actual Effort” textbox under “Fix Effort” section and includes some detail in the Fix Detail part. Likewise, when a Tester completes the Test, “Actual Effort” and “Test Detail” sections are filled by the Tester. Doing these, will leave us with a set of historical data about Bugs that can be helpful in estimating the cost and effort of future similar Bugs. Reviewing the historical data of similar Bugs can also give hints of what the issue source and the possible solution might be reducing the actual required debugging efforts.
Chapter 6

Available Features for Each User Type

As mentioned in previous chapters, Adler Test Manager is a role-based Test Manager and unlike other Test Management Tools such as Microsoft Test Manager, each user can access only the set of features designed for the user based on his/her role. On the other words, the user-type is specified when a user is logged into the Adler Test Manager during user authentication, and then, the user is directed to the features’ main page that is foreseen for that user-type. This role-specific design pattern prevents the user from accessing any available features that the user is not allowed to access and use. This makes a perfect sense because what can be the logic behind displaying a feature to a user but not allowing him/her to use it?

Adler Test Manager is designed to be used by four user-types: Administrator, QA Director, Tester and Developer. Each of these user-types is allowed to accomplish specific tasks and hence is being presented only with the features enabling him/her to perform those tasks. In the following, available features for each user-type are introduced with their actual links to associated web pages and are further discussed in more detail.

6.1 Available Features for Administrator

A user with Administrator role/user-type is the user who performs modification to the Adler Test Manager tool and is able therefore to access and modify the code that enables this test management tool to provide predesigned available features to the users with other user-types and to operate as designed.

The code that is used to display the tool’s proper web pages and allows the tool to perform correct actions in response to the user activities is stored in the GUI table differentiated by class and function columns’ name. The classes stored in the Class table are also being used for the tool operation. Therefore, a user with Administrator role needs access and modification permission to these two tables. But in order to access and modify these two tables, the Administrator does not need to use the MySQL database directly. There are features designed in the Adler Test Manager that allows the Administrator to manipulate these tables through the GUI Admin Control Panel Page. Figure 6.1 displays this web page.

As figure 6.1 implies, there are two links, namely table and list, associated with each of the database tables: Metaclasses Management for managing class table and Template Management for managing GUI table.
Figure 6.1 Administrator Control Panel Page

Figure 6.2 Template Interface Page with List View

Figure 6.2 displays the page associated with the list view of the Template table management that is displayed when the Administrator clicks the List link of the Template Management section. As you can see in this figure, there is a drop down list of the class and function pairs available in the GUI table allowing the Administrator to select a pair from the list and perform the edit and delete operation on the selected pair by clicking the relevant button. The delete button asks for confirmation before deleting the selected pair.
form the list and the edit is performed upon confirmation. Cancel button cancels the requested operation returning the Administrator back to the Control Panel page.

The Administrator can also generate a new GUI page by clicking the New button which will display the page that allows the creation of a new GUI page. On this page, the Administrator types the class and function name in the specified textboxes and then the web page code in the specified textarea. Upon attempting to save the new GUI page, the Adler Test Manager includes the required class name necessary for running the web page code and stores the new GUI code with the determined class and function name in the GUI table displaying the success message to the Administrator. If for any reason the new GUI page code cannot be created, such as duplicate class and function name pair, the system displays an error message specifying the reason allowing the user to correct it. If the cancel button is clicked, the new GUI page code will not be saved in the GUI table. Figure 6.3 displays the Create New Template page.

![Template Create New Page](image)

**Figure 6.3 Template Create New Page**

Figure 6.4 displays the table view of the GUI table management interface. As this figure implies, the user can perform the edit and delete operations by clicking the relevant button the same way like in the list view, with the difference that there are edit and delete buttons one per each row of the GUI table, each web page code, to be used and that the editing of the code is being done before clicking the associated edit button. The new button is also available on the bottom of the page to be used for generating a new GUI code. The new button on this table interface page operates the same way as the new button on the list interface page.
6.2 Available Features for QA Director

A user with QA Director Role can perform Test Case Management, Test Plan Management, Bug Management, Code Patching Management and Testing and Debugging Cost and Effort Estimation. Figure 6.5 displays the QA Director Control Panel page and the available features designed for this specific user-type.

As figure 6.5 implies, except Code Patching Management that can only be performed through the list interface, the management of other available features for a user with QA
Director role can be performed by either table interface or list interface based on the QA Director choice and preference. Each of these features is discussed in more detail in a separate section below.

### 6.2.1 Test Case Management

QA Director is one of the user-types that can perform Test Cases management. However, only a QA Director has the feature to create new Test Cases. Test Case Management can be performed through either table or list interface. If the list interface is being used, then, there is a drop down list of already existing Test Cases with Edit and View buttons allowing the QA Director to edit or view any selected Test Case from the drop down list. Clicking these buttons will direct the QA Director to the edit or view page with respect to the clicked button allowing the operation of edit or view action. Clicking the New button will open New Test Case Generation page enabling the creation of new Test Cases one at a time. Figure 6.6 displays the Test Case Management page with list view available for a user with QA Director role.

![Test Case Management Page with List View](image)

**Figure 6.6 Test Case Management Page with List View**

The Table link will display the table view of the existing Test Cases ordered by Test Case ID with an Edit and a View button to select associated with each Test Case. In order to edit or view any existing Test Case, the respective Edit or View button should be clicked and upon clicking, the QA Director will be directed to the Test Case Edit or View page depending on the clicked button, allowing the respective operation's occurrence. The New button on the neat of the page will allow creating a new Test Case by displaying Generate New Test Case page enabling the new Test Case generation. Figure 6.7 displays the QA Director’s Test Case Management page with table view.
6.2.2 Test Plan Management

QA Director is the only user-type that can manage Test Plan. This user-type has the features to create new Test Plans and edit existing Test Plans but not delete them. Similar to Test Case Management, the Test Plan Management has two interfaces namely table and list. The functionality of table and list interfaces is the same with different views. Figure 6.8 displays the Test Plan Management page with list view and figure 6.9 displays the Test Plan Management page with table view.

Once a Test Plan is created and saved, it cannot be deleted. However, the QA Director can change the Test Plan Status to inactive which will remove all that Test Plan's not tested Test Cases from the Tester's ToDo list meaning that the Test Cases of an inactive
Test Plan will not be active for testing as well. When a Test Plan becomes inactive, all its associated Test Cases become inactivate too and require no more testing.

![Test Plan Management Page](image)

**Figure 6.9 Test Plan Management Page with Table View**

### 6.2.3 Bug Management

A QA Director has both table and list interfaces for managing Bug. Figure 6.10 displays the Bug Management page with list view and Figure 6.11 displays the Bug Management page with table view. The two Bug management interfaces available to a user with QA Director role allow editing Bugs but not creating a new Bug or deleting an existing Bug. Once a Bug is created, it cannot be deleted. However, it can be switched to inactive state if not needed anymore. If no edit is required and the QA Director just wants to view a Bug detail, View button is the option that can be used.

![Bug Management Page](image)

**Figure 6.10 Bug Management Page with List View**
Editing a Bug for a QA Director means assigning it to a Developer for resolving the associated issue, assigning it to a Tester to assess the correctness of the related fix suggested by the Developer, changing the state of a Bug to inactive, defer or complete whichever applies each time.

Accessing to the Bug Repository is another feature available to the QA Director through “Repository” button. Clicking the “Repository” button directs the QA Director to the Bug Management Page enabling the edit or view operation of all existing Bugs in the database regardless of the Bug Status.

6.2.4 Code Patching Management

The code patching which can be performed from Dev Software copy to Test and Release copy, and also from an old version of the Release copy to Dev copy is the feature available only to QA Director user-type. Upon clicking the list link associated with Code Patching Management section, Code Patching Interface Page will be displayed. Figure 6.12 displays the Code Patching Management page.

Once a Developer announces changed/new class and function pairs, the announced pairs are saved in the change-announcement table. The list of these pairs is being shown in a table view on the Code Patching Page. Column “Ready To Release” identifies which class and function pair listed in the table are ready to release. There are two buttons available for code patching; “Release To Test” for patching from Dev copy to Test copy and “Release” for patching from Dev copy to Release copy. The QA Director can click any of these two buttons to perform the patching to the desired Software copy. Clicking any of these buttons will perform the patching of the “Ready To Release” class and function pairs to the appropriate Software copy.
The listed class and function pair announced by Developers can be compared to the currently in use or an old Software Release copy through using “Compare to the Current Version” and “Compare to an Old Version”. After comparing the new, by a Developer proposed code to an old version, the QA Director may convert the code to that old version in the Dev Software copy using “Get This Version” button. Then, this code version can be released to Test Software copy to be tested.

Code Patching Management is a feature designed for version control over the underlying software and is available only for a user with QA Director role. This is the Adler Test Manager built-in unique feature that allows code patching without requiring the use of any common version control software such as SVN and Git.

6.2.5 Testing/Debugging Cost & Effort Estimation

This feature allows a user with QA Director user-type to review the estimated and actual cost and effort spent on existing Bugs by clicking the respective link in this section. The main purpose of this feature is to enable estimating the cost and effort required for new bugs based on reviewing some similar previously handled Bugs from the system. Using this feature improves the precision of the testing/debugging cost and effort estimation assisting in effective project management. Figure 6.13 displays the Historical Data Display page.

In the Historical Data Display page, there are filters available for the QA Director to select from. These filters allow the QA Director to reduce the number of Bugs being displayed to only the Bugs with the desired properties for more effective estimations. Once the “Display” button is clicked, the Bugs with the selected properties are displayed in a table with the estimation helpful fields of Bugs allowing the QA Director to see the
selected Bug in full detail if needed by clicking the “View Bug Detail” button associated with the selected Bug.

![Historical Data Display Page](image1.png)

**Figure 6.13 Historical Data Display Page**

### 6.2.6 Create New User

Figure 6.14 displays the Create New User page. This page enables the QA Director to define a new user with one of the available roles assigning a unique Username (Login) and Password to be used for that new user.

![Create New User Page](image2.png)

**Figure 6.14 Create New User Page**

### 6.2.7 Change Password

Figure 6.15 displays the Change Password page which allows changing the QA Director password. Once the password is changed, a rellogin is required.
Figure 6.15 Change Password Page

6.3 Available Features for Tester

Figure 6.16 displays the Tester Control Panel Page. There are three main features available to a user with Tester role namely ToDo List Management, Bug Management and Password Management. These features are further explained in the following three subsections.

Figure 6.16 Tester Control Panel Page

6.3.1 ToDo List Management

ToDo List Management feature enables the Tester to manage his/her work load using either table or list links. The table interface displays the Test Cases that need to be tested by this Tester in a table view allowing him/her to select one and start with the testing of
the selected Test Case. The Test Cases are displayed in order based on the Test Case priority field.

The Bug table is the second table in this page that is displayed under the Test Case Table and lists the Bugs that are assigned to the Tester after the proposal of the associated Bug fix. The Bugs are ordered based on their priority and the Tester can either view or edit the Bug by clicking the related button present for each Bug. Figure 6.17 displays the Tester ToDo List Page with table view.

![Figure 6.17 Tester ToDo List Page with Table View](image)

Clicking the Start button associated with each Test Case will direct the Tester to the Test Case Testing Page allowing the Tester to start testing that Test Case. Clicking an Edit button will display the related Bug’s Edit Page where the Tester can perform update to the Bug while a View button just displays the Bug information without editing ability.

Unlike the table interface that displays Test Cases and Bug List with values of some relevant fields, the list interface displays only the title of the Test Cases and Bugs listed based on their priority. Thus, Test Cases with higher priority appear on the top of the drop down list encouraging the Tester to perform the testing operation starting from the top of the list all the way down to the bottom. The same thing is true for Bugs. Testers are encouraged to work on Bugs in the order of their priorities. Figure 6.18 displays the Tester ToDo List Page with List View.
Clicking the Detail button will direct the Tester to the View Test Case Detail Page displaying the selected Test Case information and Steps in order allowing the Tester to review the Test Case detail. Figure 6.19 displays the View Test Case Detail Page.

Clicking the Start Testing button present on the top right side of the page will direct the Tester to the Testing Page where the Tester can perform the testing. In this page, each Test Case Step is displayed one at a time in order and can be passed by clicking Pass button or failed by clicking Fail button. If a Step is passed, the next Step will pop up until the last Step is passed and this is the time when the Test Case is “Pass” and “Complete”. However, if a Step is failed, “Create New Bug” button will pop up allowing the Tester to create and save a new Bug associated with the failed Test case and Test Case Step. In this
case, the related Test Case along with the Steps up to the failed Test Case Step for regenerating the Bug will be included in the created Bug upon saving the new Bug.

6.3.2 Bug Management

Bug Management feature which is available in both table and list view, allows the Tester to Edit/View not completed Bugs regardless of the Bug state or whom it is assigned. This is because the Tester might add some useful information that can ease the understanding of the underlying issue leading to easier and faster Bug fix.

As mentioned before, Tester can create a new Bug during Testing a Test Case when a Test Step is marked as Fail. The Create New Bug button that becomes available once a Test Step is failed, directs the Tester to the Generate New Bug Page. However, a Tester can create a new Bug directly without requiring failing any Test Case Step using the New button of the Bug Management feature available on the Tester Control Panel Page.

If a Bug is created directly from using Bug Management feature, the Tester will require explaining how to reproduce the Bug in the “Bug Reproduction Steps” section. In the “Bug Detail” section the Tester can then add some more detail that might help regenerate the Bug and debug the code by the Developer. The “Bug Detail” section can also be used by the QA Director to determine the severity level of the Bug.

Besides editing, viewing incomplete Bugs and generating new Bugs, accessing to the Bug Repository is also available for a user with Tester role through the Repository button available in the Bug Management Page. The Tester can access the Bug Repository and review a Bug’s information when needed. Figure 6.20 displays Bug Management page with table view and Figure 6.21 displays Bug Management page with list view.
6.3.3 Password Management

Password Management is another feature available to the Tester. Clicking Change Password link will direct the Tester to the Change Password Page which has the exact same functionality and user interface as this feature that is available to the QA Director.

6.4 Available Features for Developer

Developer is the user-type that deals with Bugs, under Dev Software code and Code Change Announcement. This user-type has therefore the features for accomplishing these tasks available on the Developer Control Panel Page. Figure 6.22 displays the Developer Control Panel Page and available features for Developer.
6.4.1 ToDo List Management

The ToDo List contains Bugs that need to be resolved in the underlying Dev Software Application assigned to the signed in Developer. The listed Bugs in the ToDo List are ordered based on their priority and are available in table and list interface accessible through two links; Table and List respectively.

Table interface displays all the assigned Bugs with some information such as the Bug title and its severity in a table view allowing the Developer to select one at a time and start with debugging the related code. Clicking the Edit button associated with the selected Bug will display the Bug in all detail allowing the Developer to read the related information and grasp the underlying issue. Clicking the View Button will display the selected Bug information without editing capability. This is useful when a Developer wants to review a Bug in detail without the intention of modifying any of the related information associated with the Bug. Figure 6.23 displays the Developer ToDo List page with table view.

<table>
<thead>
<tr>
<th>Bug_ID</th>
<th>Title</th>
<th>State</th>
<th>Assigned_to</th>
<th>Area</th>
<th>Feature</th>
<th>Priority</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>this is another sample bug</td>
<td>Dev</td>
<td>David Trash</td>
<td>Template Management</td>
<td>150</td>
<td>2</td>
<td>Low</td>
</tr>
<tr>
<td>6</td>
<td>let th dud</td>
<td>Dev</td>
<td>David Trash</td>
<td>Template Management</td>
<td>150</td>
<td>2</td>
<td>Low</td>
</tr>
</tbody>
</table>

Figure 6.23 Developer ToDo List Page with Table View

List interface on the other hand, displays the list of Bugs’ title in a list ordered by Bug severity to the Developer allowing him/her to pick one from the list. By clicking the Edit button, the selected Bug from the Bug list will be displayed in full detail to the Developer allowing the Developer to perform Edit operation. Clicking View Button will display the selected Bug information without allowing any modification. Figure 6.24 displays the Developer ToDo List Page with List View.
Once the code is debugged and a quick software run by the Developer shows the expected result and the Bug is fixed, the Developer puts notes regarding the fix in the Fix Detail section of the Bug on the Bug Edit Page and saves the Bug. In the next step, the Developer is required to announce the code changes using the Code Management feature.

6.4.2 Bug Management

Bug Management feature available to a user with Developer role works the same way as for Tester. A Developer can create a Bug and edit/view existing Bugs like a Tester directly using Bug Management feature. Table link will display the table interface and the List link will represent the list interface allowing the Developer to perform both creating a new Bug and editing/viewing existing Bugs one at a time exactly the same way that a Tester can do.

6.4.3 Code Management

Once the code is changed and the changed code works the way it should, the Developer should announce the class and function pairs that are changed. Code Management feature provides the Developer with Code Change Announcement link clicking which will direct the Developer to the Code Change Announcement Interface Page where code changes can be announced. Figure 6.25 displays the Code Change Announcement page.

Figure 6.24 Developer ToDo List Page with List View
The top section that allows change announcement to the GUI table has a drop down list containing the existing Bugs’ name assigned to the Developer for fixing. The Developer can select the desired Bug from the list. Since each Bug is associated with a project, upon selecting a Bug and clicking the Start Button, the GUI table of the project related to the Bug is identified and a list of the project’s class and function pairs pops up. The popped up class and function pairs list is ordered by class and function. Figure 6.26 displays the Code Change Announcement page with the class and function pairs drop down list.

Once the class and function pairs list is available, the Developer can select the class and function pair(s) modified to fix the related Bug one at a time followed by clicking the Announce button each time. This button will record the name of the selected class and
function pair in the change-announcement table. If the pair already exists in the table, then, the Developer will receive a warning message informing that the pair already exists in the table. The class and function pairs are announced one at a time.

The bottom section present in this page is a list of class and function pairs announced by the Developer. This table displays only the announced class and function pairs that are not released by the QA Director. Once the QA Director releases a class and function pair, that pair will be removed from this list and will not be displayed any more. The Delete button associated with each class and function pair enables the Developer to remove that pair from the Code Change Announcement table.

Once all the code changes regarding a Bug are announced, the Developer can go back to the Developer Control Panel Page and add some detail regarding the performed code fix in the Fix Detail section of the Bug and save the Bug. Upon doing it, the Bug will be removed from the Developer ToDo List and the Developer can start to work on the next assigned Bug.
Chapter 7

Adler Test Manager Unique Characteristics

Adler Test Manager is a unique test management tool that not only assists in managing software manual testing, but also provides version control functionality and supports project management. Beside features for performing manual software testing and keeping track of the testing status along with features to create, assign and track Bugs, Adler Test Manager has a built-in version control mechanism that allows code patching using relational database tables. Moreover, this tool supports project management with its task management and testing/debugging cost and effort estimation features. Below in this chapter, the unique characteristics of Adler Test Manager are compared to some available tools with similar functionalities.

7.1 Test Management Unique Characteristics

Unlike some testing tools such as Microsoft Test Manager that displays the whole active Test Plan with a list of the assigned Test Cases to different Testers allowing any user to access and even to modify any Test Case in the Test Plan regardless of who the assigned Tester is, Adler Test Manager is a role based tool that has a different user interface for each user type (role). Microsoft Test Manager allows all users to access all pages with some features disabled, yet Adler Test Manager displays only the features and pages that the signed in user type is authorized to use. The role based methodology used in the Adler Test Manager design has provided each user type with a different user interface with only the features that satisfy that user’s needs to perform the duties he/she is assigned to.

For instance, in this testing tool, only a user with QA Director has the feature to access the Create New Test Plan page and to view and modify the active Test Plan. Also, each Tester can access only the Test Cases that are assigned to him/her for testing through the active Test Plan in the Tester ToDo list page. If a user has Developer role and his/her duty is to fix Bugs and enhance the software, there’s no point to allow this user to access Test Plan. A Developer can only view a specific Test Case regardless of whether it’s in the active Test Plan or not. Adler Test Manager checks the user role to ensure that the user has the authorization to access the requested service/page before allowing the user to access a page and use a feature. If any user is identified as unauthorized, the Adler Test Manager logs the user out displaying the login page asking for relogin.

7.2 Version Control Unique Characteristics

Unlike other version control tools such as SVN and Git that control and track changes in files, Adler Test Manager tracks source code revisions using a relational database. Since each web page in the software under test is corresponding to a row in the Template
database table with unique class and function set, the built-in version control mechanism performs tracking and control over changes in code associated with each Template table row. This mechanism stores the old code in the Template_backup table associating the create date to the code enabling the retrieval of any version of the code accessible using the create date.

Likewise, when it comes to patching code from one software copy to the other, Adler Test Manager releases only the changed/enhanced code instead of patching the whole software code. This means that only code associated with the class and function sets announced as changed/enhanced are being patched from one software copy to the other after reviewing and approval of the changes. Thus, the version control is on the Template table rows each of which represents one web page instead of keeping the control over files.

Moreover, when switching to an old version, just one web page code stored in a Template table row is reverted into an older version instead of a file. Therefore, using the built-in Adler Test Manager version control mechanism provides control over code stored in the database and code versioning is being applied to each web page code separately and there’s no whole software code versioning leading to save a lot of memory space by not saving unchanged code if we were to create multiple versions of the entire software code.

7.3 Project Management Support Unique Characteristics

While project management tools allow the user to schedule and assign tasks and to allocate resources without providing support for tasks completion cost and effort estimations that are close to the actual cost and effort, Adler Test Manager assists in performing testing and debugging cost and effort estimations by allowing the storage and retrieval of historical data. This amazing tool not only provides a user with QA Director role to create Test Plans and assign Test cases and Bugs to Testers and Developers, it provides the required features to store Bug details such as primary estimated efforts and actual efforts and the features to retrieve these data when performing estimations for similar Bugs in similar projects or project areas.

Furthermore, the fix detail and test detail fields can be useful in providing hints on how to handle similar Bugs leading to reduced development efforts. This is because the same issue source might be the root cause of the Bug introduction to the system knowing which can lead to easy and fast fix if this is the case. Beside this, the severity level of similar Bugs can help in allocating the most proper human resource to handle each Bug. Storing and retrieving historical data functionalities designed in Adler Test Manager has made this tool a unique test manager that supports project management.
Chapter 8

Conclusion and Future Work

Adler Test Manager is a web based manual testing tool that assists in maintaining the quality of software products. It is a software package that not only allows Test Case and Test Plan generation and maintenance allowing them to be used for testing underlying software products, but also keeps track of Bugs and code patching through its built-in mechanism. Using this Test Management tool does not require any version control or Bug tracking software application as the necessary functionality is implemented in the tool using MySQL relational database.

Unlike many Test Management tools such as Microsoft Test Manager, this manual testing tool is role-based and provides each user with the set of features designed for that specific user based on his/her user-type so that all the accessible features to a user can be used with no exception or limitation.

Furthermore, this stand-alone testing tool supports project management by providing the management with the Bug related information history that can be used in estimating testing/debugging costs and efforts with a higher precision. Reviewing Bug history can give the project manager a good assist in making testing/debugging cost and effort estimations closer to the actual values that will take place. More precise estimations can help project managers in a more successful task planning and scheduling.

8.1 Future Work

Adler Test Manager is a manual testing tool that requires the creation and maintenance of Test Cases and Test Plans. It creates a historical data collection regarding Bugs that can be used for project management. However, this management tool does not assist in planning and scheduling software development tasks unless they are related to Bugs. A future work can be extending the tool to be also a project management tool that can be used for planning and scheduling development tasks and estimating development costs keeping track of the tasks progress, alarming the project manager for the tasks that are behind the schedule and the chances are high that they will slip the schedule, allowing rescheduling the remaining tasks if there is a task slippage to manage the situation and recalculating the project costs.

Extending the Adler Test Manager to include automated testing feature is another aspect of the future work that might be done to have a more flexible and user friendlier testing tool. Some people prefer automated testing over manual testing these days and including this feature may motivate more people to use and benefit Adler Test Manager tool.
References


Appendix A: Database Schema

**Template table:** this table holds the PHP code for web pages that are being displayed to the users. The combination of class and function is the primary key in this table. Each time a user requests for a web page, class and function parameters are set automatically and the associated code is fetched and evaluated to display the requested web page to the user. The req-classes field holds the class-id of the classes required for each web page code.

**MetaClass table:** this table holds required classes for web page code proper evaluation and display. The class-id which is the primary key in this table is connected to the req-classes in the Template table.

**Project table:** this is the table that holds information about the projects under test. This information is available to other tables through project-id primary key.

**Test Case table:** Test Case information is stored in this table. This information can be accessed through tc-id primary key.

**Test Case Step table:** this table holds the Test Case Steps for each Test Case. The Steps are associated with each Test Case through tc-id foreign key.
Test Plan table: Test Plan information is stored in the Test Plan table with primary key plan-id. The project_id foreign key specifies the project associated with each Test Plan.

Test Case Plan table: this table holds the status info of the Test Cases assigned to Testers for testing in Test Plans. The foreign key plan_id can be used to fetch the plan info associated with each Test Case Plan. The tc-id is the foreign key that points to the related Test Case information in the Test Case table. The Test Case Plan table is connected to the Bug table through its bug-id foreign key.

Bug table: this table keeps data for the proposed Bugs associated with testing Test Cases. bug-id is the primary key in this table. Bug table is connected to Test Case table through its tc-id foreign key. Also, the assigned-to, tester and developer fields are connected to the user-id primary key in the User table.

Code Change table: this table holds the class and function name of the code that are modified to fix each bug. The bug-id foreign key specifies the Bug that is associated with each modified/changed code. The developer field in this table is connected to the user-id primary key in the User table.

User table: this is the table that stores information about the users including the users’ login, password and role.