

# AUTOMATED REGRESSION TESTING FRAMEWORK

N GNANASEKARAN  
VINEET BANGA

## Table of Contents

1. INTRODUCTION	1
2. AUTOMATED REGRESSION TESTING FRAMEWORK	1
1. Approach	1
2. Architecture	2
3. Configuration file relates to Framework Settings	2
4. Benefit to execute from SQL Server Database over excel sheets	2
5. Web Interface to create/maintain Test Plans and Test Cases	3
6. Report Design/Template	4
7. Templates	4
8. Test Execution Flow	5
9. GUI Map verification	6
3. TESTING PROCESS	7
4. OTHER DETAILS	8
5. SUMMARY	8

## Introduction

There has been many automated software testing process existing and being developed by huge number of testing professional. Automated testing slowly becoming a necessity as the demand for low turn around time and high quality.

There are different regression testing tools (Win runner, Robot, QA Run) based on GUI and type of application (Client Server, Web application). The major focus or issue lies in implementation of these off the shelf product in a customized way. This approach is one step in the direction of recording and allows you to create test cases. It provides a guide to write test cases in a customized way. It uses the keyword and commands generated based on event triggered on each field, form, page in the application. This is one of a cost effective approach for automated testing solution.

The purpose of this paper is to provide a clear understanding of what is required to implement automated testing successfully. We had provided details steps involved in generation, standards followed for creation of the framework that can be used with any regression-testing tool. We had provided the templates, execution flow reporting used and Testing process followed for implementation of this framework.

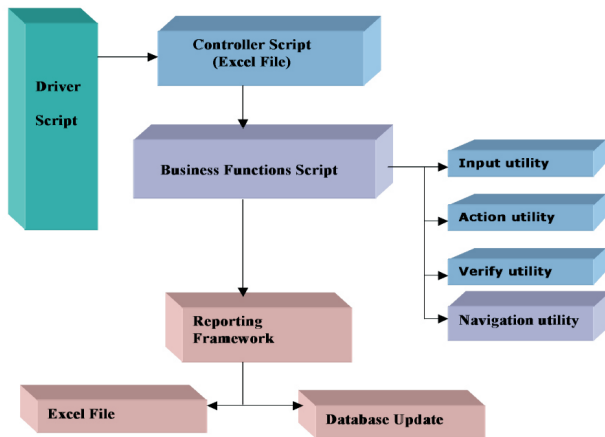
## 2. Automated Regression Testing framework

### 1. Approach

The Regression Testing framework is based on the "Test Plan Driven" method (a.k.a "Key-word driven). In this method, the entire process is data-driven, including the functionality. The Actions/Methods (keywords) control the processing. In this approach a web interface is used to generate the test plan and test cases. Which are then stored in the form of tables in a Database.

## 2. Architecture

Following block diagram shows the components of the regression-testing framework:



The Framework constitutes of the following main components, which are explained in the sub-sections given below:

- Driver Script
- Controller Script
- Business Function Script
- Utility Scripts
- Reporting Framework

1. Driver Script: The Driver script is the main startup script. It performs initialization and then calls the Controller Script.

2. Controller Script: The Controller script calls the Generic scripts associated with Business function to perform the Test Case actions and verifications.

3. Business Function Script: The Business Function script would perform the following tasks:

- This script performs application specific business testing based on the testing scenarios as given in the test cases spreadsheet by calling appropriate utility scripts to perform specific actions

4. Utility Scripts:

The Utility scripts would perform the following tasks:

- Utility scripts would perform any particular task based on the objects active in the open window e.g.
  - Input utility scripts would let the user input data in controls
  - Action utility scripts would perform any actions on the windows or controls
  - Verify utility scripts would validate the values retrieved from the data sheet with the values in the controls.
  - Navigation utility scripts would provide navigation related functionality to the user

5. Reporting Framework:

The followings are the importance for the Reporting Framework:

1. Reporting Results: The Reporting framework would provide generic methods to generate test results/reports for the performed tests. The reports would be stored in a standard Excel format.

2. Test Entities Maintenance: The Reporting framework would provide a mechanism to maintain the list of all the Test Entities (i.e. Test Cases), their relation and their corresponding version history. The Test Entities would be read from the corresponding folders on the hard disks to retrieve the following information:

- Name (File name)
- Description
- Version History
  - Version # (Major & Minor)
  - Date
  - DeveloperID
  - Change Description

3. Configuration file relates to Framework Settings

A Configuration file would be created to define the settings related to the Framework. All the parameters that help the framework to execute are specified in the Configuration file along with their values. The Framework reads the value of these parameters on execution and maintains that settings.

E.g. "Execute using XLS or Database (Specify XLS for Excel or DB for Database).

4. Benefit to execute from SQL Server Database over excel sheets

The Framework provides the benefit to create/execute/maintain the Test Entities (Test Plans and Test Cases) from standard Excel format or SQL Server database. If the framework uses the Excel format then a folder location where test entities reside needs to be specified in Configuration file based on which the Framework reads the test plans / test cases.

Using SQL Server as database a Web Interface(PHP) is created that stored and retrieved the test plans / test cases details from the database. The Web Interface provides the following benefits:

- Global repository where data stores in the SQL Server database
- An intelligence mechanism that helps the users to create the test cases by automatically reads windows/objects from the selected GUI Maps file
- Provide the user authentication and security administration feature
- Provide the criteria where users can schedule the execution of the framework.

5. Web Interface to create/maintain Test Plans and Test Cases

A Web Interface (using PHP and PEAR) created to maintain test plan / test case in SQL Server tables. This application will be invoked from web browser e.g. Internet Explorer. Following sample depicts the screen layouts of application:

[Test Case]

Name

Description

Deprecated (Y/N)

Dependent Test Case(s)

Version Number  Date  Developer ID

Change Description

Window	Field	Label	Row	Value	Method	Test Case Description
winPreLogin	btnOK				ButtonClick	Click on the 'OK' button in the Pre Login window.
winLogin	txtUserName			User1	TextInput	Enter 'user1' in User Name textbox of the Login window.
winLogin	txtPassword			init	TextInput	Enter 'init' in Password textbox of the Login window.

Test Case Screen

[Test Plan]

Name

Description

Deprecated (Y/N)

Version Number  Date  Developer ID

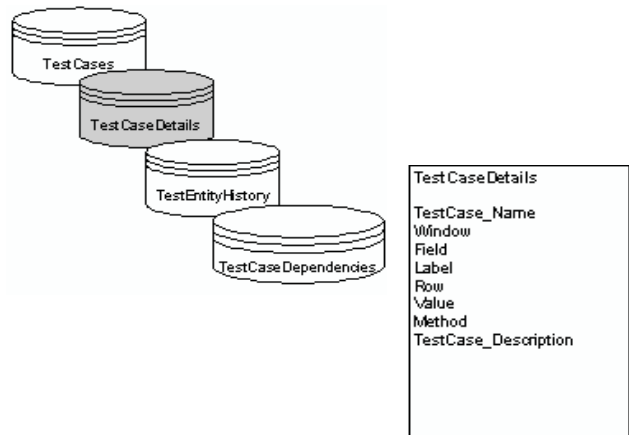
Change Description

Test Case / Test Plan Name	Description	Module Name
Test Case_SignOn	Perform Sign On	Common

Test Plan Screen

While creating a Test Case/Test Plan', following SQL Server tables will be updated respectably:

- Test Case  
[TESTCASES]  
[TESTCASEDETAILS] - a new table for Test Case's information  
[TESTENTITYHISTORY]  
[TESTCASEDEPENDENCIES]
- Test Plan  
[TESTPLANS]  
[TESTPLANDetails] - a new table for Test Plan's information  
[TESTENTITYHISTORY]



### 6. Report Design/Template

The following figure presents the schema design to support the results of the Reporting framework.

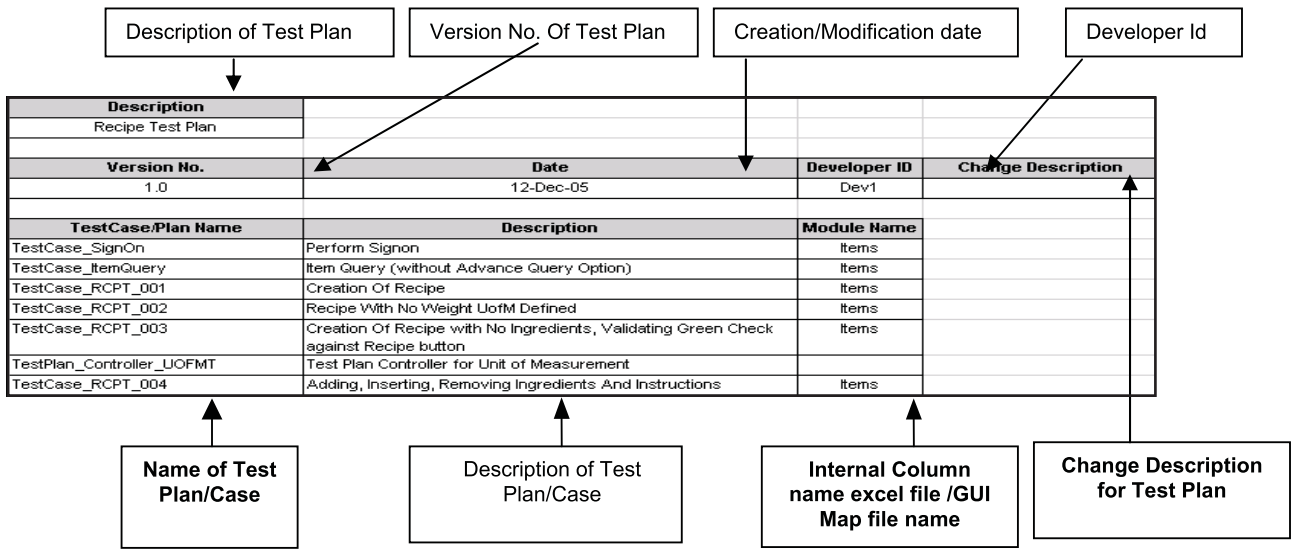
Test Results										
Execution Date/Time	Application Name	Application Version	User ID	TestPlan Name						
Oct 27 2005 21:06:19	CBORD FMS	4.0.140 (Build 5320m)	user1	TestPlan_Controller						
Test Case Name	Window	Field	Sub-Field	Row	Value	Method	Test Description	Result	Message	Remarks
TestCase_SignOn	winPreLogin	btnOK				ButtonClick	Click OK in the Pre-Login window	Pass	Button Click performed successfully for - btnOK	
TestCase_SignOn	winLogin	txtUserName			user1	TextInput	Enter a correct userid	Pass	Input successful for Value- user1	
TestCase_SignOn	winLogin	txtPassword				TextInput	Enter a correct password	Pass	Input successful for Value- init	
TestCase_SignOn	winLogin	user1	init			SpecifyFMSUserName	To get User Name from which the FMS application logged	Pass	User's credentials stored	
TestCase_SignOn	winLogin	btnOK				ButtonClick	Click OK in the Login window	Pass	Button Click performed successfully for - btnOK	
TestCase_SignOn	winPostLogin	btnOK				ButtonClick	Click OK to close the Post Login window	Pass	Button Click performed successfully for - btnOK	
TestCase: TestCase_SignOn Passed										

Excel File name format - <TestCase Name> DDMMYYYY\_HHMMSS.XLS

### 7. Templates

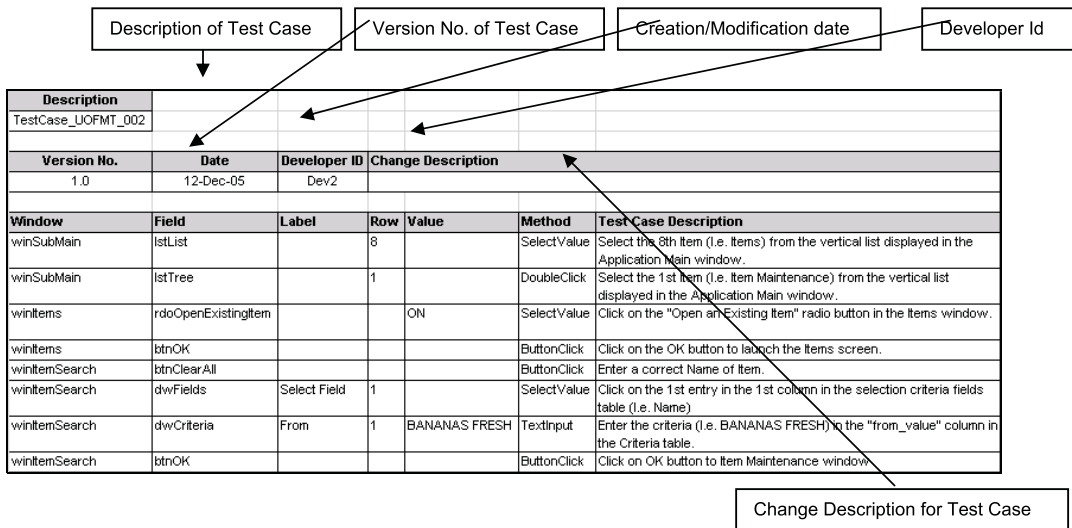
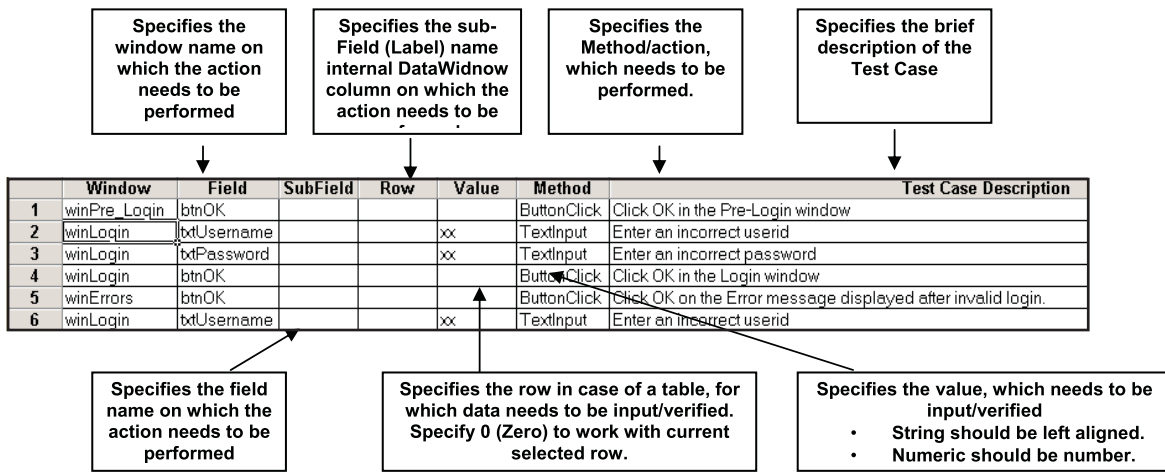
Test Plan

The Test plan is used to determine the sequence of the top-level test scenarios or test cases, which need to be executed. The figure given below presents the sample test plan and describes the various columns available in the test plan. Each row in the test plan documents a test case/scenario, which is specified by the name of the Business Function test case.



Generic Business Function Test Case

The business function test case template is a generic test case template, which can be used to prepare automated data-driven test case for any test scenario. The figure given below presents the sample test case. The figure also describes the various columns available in the generic business function test case. Each row in the Business Function script contains an action/method, which needs to be executed by the script. Along with the action/method, the row contains information about the window and field on which the action has to be executed and the corresponding value on which it needs to act upon.



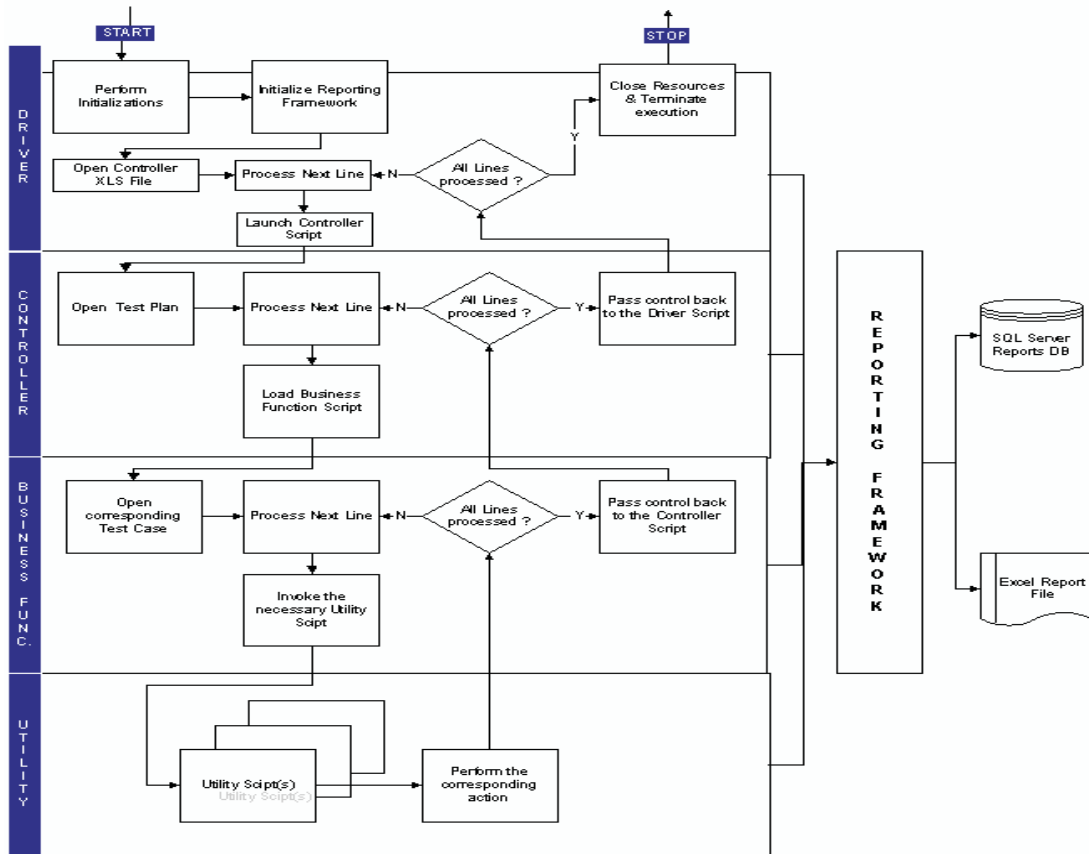
8. Test Execution Flow

This section describes how the Regression Testing framework would work. The following are the sequence of steps that would be carried out when the framework would be used to carry out regression testing:

- The Driver script (TSL script) would first be loaded and executed. The Driver script would carry out the necessary initializations for the framework, Reporting framework. The Driver scripts would read test plans name and then would pass each test plan to controller script for further execution.
- The Controller script (TSL script) controls the sequence of execution of the test cases/scenarios through an associated test plan/data. The test plan is created in an excel spreadsheet or using the web interface. Each row in the test plan documents a test case/scenario, which is specified by the name of the Business Function test case, which needs to be executed by the Controller script along with a brief description of the test case.
- The Controller script reads the associated test plan starting from

the 1st row. The Controller upon reading the name of the test case from the test plan spreadsheet then loads GUI map file. After that invokes the Business Function script passing it the name of the associated Business Function test case.

- The Business Function script opens the test case spreadsheet, which was passed as parameter from the Controller script. Each row in the Business Function script contains an action/method, which needs to be executed by the script. Along with the action/method, the row contains information about the window and field on which the action has to be executed and the corresponding value on which it needs to act upon.
- The Business function script upon reading the action/method from the test case invokes the corresponding utility script passing details about the window name, field/sub-field name and value. The utility script executes the desired action and returns the control back to the business function script, which then starts processing the next in the test case. The business function script continues to process each row in the test case till a blank row is



encountered upon which the control is transferred back to the Controller script.

- After the Business function script completes execution, the controller script starts processing the next row in the Test plan. The Controller script continues to process each row in the test plan till a blank row is encountered upon which the control is transferred back to the Driver script. The Driver script then performs any completion actions like unloading any compiled modules, closing/terminating the Reporting framework, etc.
- During execution of the framework, in case if any test case is stopped, the framework would read the dependent (child) test cases information and would store into an array along with the parent test case.

**9. GUI Map verification**

To perform GUI Map verification between various builds of the

application, the framework would provide method(s) to read the specified window of the FMS application. This method would traverse through all the available controls/objects contained in the specified window and would compare the same with the GUI map file.

The user can specify the sequence of steps to traverse through the entire application in a test case (as per the existing test case template). Besides containing the sequence of steps for traversing, the test case would also provide data/steps to enable/activate certain disabled elements. The GUI verification test plan/cases would be prepared separately.

E.g. the following test case depicts how, while traversing the application, checkpoints can be placed in the test case for performing GUI verification.

In this example, a call to a method named "verifyGUIMap"

Window	Field	Sub-Field	Row	Value	Method	Test Case Description
winPreLogin	btnOK				ButtonClick	Click OK in the Pre-Login window
winLogin					verifyGUIMap	Verify window/object of Login window
winLogin	txtUserName			user1	TextInput	Enter a correct userid
winLogin	txtPassword			init	TextInput	Enter a correct password
winLogin	btnOK				ButtonClick	Click OK in the Login window
winPostLogin	btnOK				ButtonClick	Click OK to close the Post Login window

Window	Field	Sub-Field	Row	Value	Method	Test Case Description	Result	Message
winPreLogin	btnOK				ButtonClick	Click OK in the Pre-Login window	Pass	Button Click performed successfully for - btnOK
winLogin					verifyGUIMap	Verify window/object of Login window	Fail	New control (Push button) found
winLogin	txtUserName			user1	TextInput	Enter a correct userid	Pass	Input successful for Value - user1
winLogin	txtPassword			init	TextInput	Enter a correct password	Pass	Input successful for Value - init
winLogin	btnOK				ButtonClick	Click OK in the Login window	Pass	Button Click performed successfully for - btnOK
winPostLogin	btnOK				ButtonClick	Click OK to close the Post Login window	Pass	Button Click performed successfully for - btnOK

instructs the framework to perform GUI verification on "winLogin" window and then report the comparison/verification results (as shown below). This report would be generated for GUI verification only.

### 3. TESTING PROCESS

#### GUI Map preparation

When WinRunner runs tests, it simulates a human user by moving the mouse cursor over the application, clicking GUI objects and entering keyboard input. Like a human user, WinRunner must learn the GUI of an application in order to work with it. WinRunner does this by learning the GUI objects of an application and their properties and storing these object descriptions in the GUI map.

When WinRunner learns the description of a GUI object, it looks at the object's physical properties. Each GUI object has many properties, such as "class," "label," "width," "height", "handle," and "enabled". WinRunner, however, learns only a selected set of these properties in order to uniquely distinguish the object from all other objects in the application.

WinRunner uses a logical name to identify each object: for example "Print" for a Print dialog box, or "OK" for an OK button. The logical name is actually a nickname for the object's physical description. The physical description contains a list of the object's physical properties: the Print dialog box, for example, is identified as a window with the label "Print". The logical name and the physical description together ensure that each GUI object has its own unique identification.

The user/tester would first need to create a GUI map by using the GUI Map Editor to learn the properties of an individual GUI object, window, or all GUI objects in a window. If the GUI of the application changes during the software development process, the tester can use the GUI Map Editor again to learn individual windows and objects in order to update the GUI map.

For creating the GUI Map File, the tester would need to teach WinRunner the information it needs about the properties of GUI objects by clicking the Learn button in the GUI Map Editor to learn the properties of an individual GUI object, window, or all GUI objects in a window. The tester should however use User-Friendly logical names while creating the GUI map file so that it is easier to correlate to the actual fields/windows while developing the test plans.

To improve performance, the tester should use smaller GUI map

files for testing your application instead of one larger file. The tester should divide the application's user interface into different GUI map files by window or in another logical manner e.g. module wise GUI Map file.

#### Test Plans preparation

After preparing the GUI maps for the screens for which testing needs to be performed, the corresponding Test plans need to be prepared. The Test plans need to be prepared using the web interface defined in earlier sections.

#### Test Plan preparation

The Tester would need to first identify at a high level the various test scenarios which would need to be automated and executed. These scenarios would then need to be generated using the web interface in a Test plan format. Each Test scenario would correspond to one row in the Test plan and would provide the following details:

- The name of Test Case, which needs to be executed for the Test scenario.
- The name of internal column/GUI map file. The name of both files should be (Internal map/GUI map) same for each module.

#### Business Functions Test Cases preparation

For each of the test scenarios identified in a Test plan, the tester(s) would need to prepare individual test cases to be generated using the web interface as per the template discussed in the earlier sections. Each row in the Business function test case would correspond to a test case specifying an action that would need to be performed on a specific window and field and would contain the following details:

- Window: Name of the window on which the action needs to be performed. Please note that the name of the window should exactly be the same as defined in the corresponding GUI map file, otherwise the Testing framework would not be able to find the same. Therefore, it is important to provide a user-friendly logical name, which maps to the actual physical window name so that the tester can easily correlate to the actual window.
- Field: Name of the field on which the action needs to be performed. Please note that the name of the field should exactly be the same as defined in the corresponding GUI map file, otherwise the Testing framework would not be able to find the same. Therefore, it is important to provide a user-friendly logical name, which maps to the actual physical field name so that the tester can easily correlate to the actual field on the screen.
- Row: Number of the row on which the action needs to be

performed. Since the application contains many screens, which comprise of a tabular/grid like controls, the Row column in the Test case template can be used to specify the row number on which the action has to be performed.

- Value: The actual value, which needs to be input or verified needs to be specified in this column. For some of the methods, the value column would contain multiple values separated by a pipe (|) symbol.
- Method: Name of method/action, which needs to be performed.
- Description: The Description is a textual brief description of the test case being documented. Please note that the description is optional i.e. the Testing framework does not process it. However, including the description increases the readability of the Test case.

## 4. Other Details

### Advantages

This method has following advantages:

1. Flexibility in terms of:
  - Data
  - Flow control
  - Re run flow multiple times
2. Low Maintenance Effort of scripts (One time effort).
3. No WinRunner Scripting knowledge required for majority of the users for using the framework.
4. Only one WinRunner license required for running the scripts from a single machine. The users/testers can develop the test plans/data without using WinRunner.
5. Testers can easily generate test plan since the test plan follows a logical flow of actions and can be easily created in Excel or by Web Interface.
6. One Data template for the entire test plans. Therefore, the testers need to understand only one template.
7. The Test Plan can also be written in Spreadsheet format containing all input and verification data. Tester needs to write this once, which can be reused with the test plan name.

### Disadvantages

1. Initial development of the function and utilities are costly and timeconsuming activity, which any developer can do after analyzing the application. This is applicable to most of the GUI based regressiontesting tool.
2. Tester needs to learn "Key Words", when & where to use it. The framework generates how to use these key words in a systematic way. Initially it could be time-consuming, and may have some impact on development of this Test Plan.

## 5. Summary

- Identify clear goals and reasonable expectations as to what can and what cannot be accomplished with automated testing.
  - Identify the risks of failure and time required to generate framework
  - Identify test cases that can be automated which may include both simple and complex scenarios so that maximum coverage is attained with minimum number of test cases.
- Identify the acceptance criteria for automated testing framework that is require to be generated. Collect and understand the requirements for successful with automated testing
  - Optimal usage of the tool
  - Effective usage of the tool
  - Regression test cases need to be identified before automation. Any undocumented procedure cannot be automated at run time. There must be:
    - Test cases with details steps followed and expected results documented.
    - A test environment with a recreated at any point
- Adopt a simple and cost effective methodology.
  - In long run record/playback is too costly to maintain and is effective in case of migration of backend application.
  - Functional Decomposition method is useful but costly too.
  - Test Plan driven method is the one of the cost-effective approach with:
    - A minimum number of automated scripts
    - Few technical personnel
    - Minimal training for the manual testers
    - Easily readable Test cases for both automated & manual testing