# Test Automation Guidelines

## BEST PRACTICES in Test tool Automation

### 1. Definition of Tests

As a prime entry point defining the test needs a idea to classify the scripts into finer elements of functions each contributing the various aspects of automation Techniques. Looking into this perspective the elements of the Automation Script would require the Record Play Back techniques, details of the application as better understood as Objects in tools, execution of Business Logic using loop constructs, and the test data accessibility for either Batch Process or any Back end operations. Ultimately we need this entire salient features to function at the right point of time getting the right inputs. To satisfy these criteria we require a lot of planning before we start automating the Test Scripts.

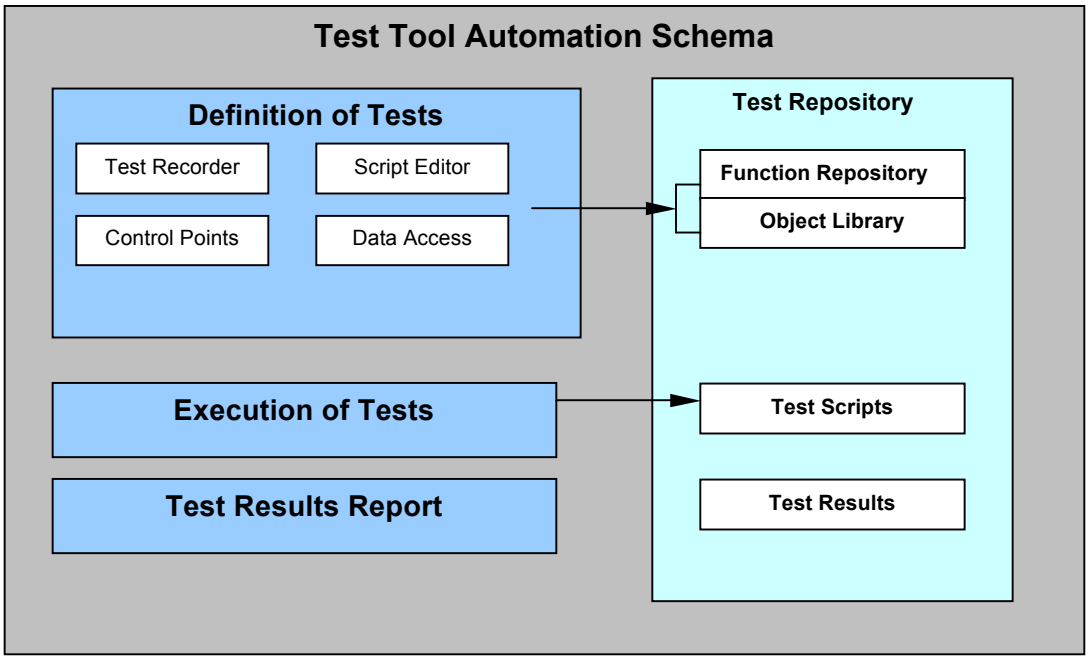



**Fig: 1**

#### 1.1 Test Recorder

##### 1.1.1 Object Vs Actions

In automation tools the Test Recorder is of two modes Object based and Action Mode. It requires a meticulous but yet a simplified approach on which mode to use. Though it is inevitable to avoid Action mode, it is still used for many at TE Based applications. As a best practice the object based is widely accepted and mandatory mode of operation in Test Automation. To the extent possible we will avoid Action based functions and stick on the object mode of operation.

##### 1.1.2 Generic Test Environment Options

Some common Settings we need to set in General Options:

1. Default Recording Mode is Object mode

2. Synch Point time is 10 seconds as default
3. When Test Execution is in Batch Mode ensure all the options are set off so that the Batch test runs uninterrupted
4. In the Text Recognition if the Application Text is not recognizable then the Default Font Group is set. The Text Group is identified with a User Defined Name and then include in the General Option.

#### 1.1.2 Test Properties

1. Every Script before recording ensure that the Test properties is in Main Test with the defaults
2. Do not entertain any Parameters for Main Test
3. It is not a good practice to load the Object library from the Test Options (if any). Rather the Object library is loaded from the Script using the suitable tool commands. This would actually avoid the hidden settings in the Script and also the ease of Setting the Object library Load and Unload can be better done dynamically in the Test Script rather than doing it manually every time the Test Suite is ran.

4. Ensure the Add-ins is correct from the Add-ins tab.

### 1.2 Script Environment

The basic idea of setting the Test Bed is that the Test Suite must be potable and can readily be ran in any environment given the initial conditions. For this to happen, the automation tool supports a lot of functions to evolve a generic methodology where we can wrap up the entire built-ins to run before the Test Suite start executing the Script. In other word the fashion of organizing the Test Scripts remain in the Test automation developer's mind to harbinger the issues and hurdles that can be avoided with little or less of programming.

#### 1.2.1 Automation Inits ()

Common Functions that get into Initialization Script are

1. Usage of built-in commands to keep the test path dynamically loaded. Options to rule out the possibility of Test Path Definitions
2. Close all the object files and data files in the Initialization Script
3. Connection to the database should be done in the Inits Script
4. Always Unload and Load the Object library, and it should be done only in Inits Script.
5. Define all the "public" Variables in the Inits Script
6. Establish the db connections in the Inits Test Script

#### 1.2.4 Test Scripts Elements:

Prior to the development of Test Scripts the fashion of arranging the Test Scripts needs a proper planning. Lets look at few inputs on arranging the Test ware.

| Test Ware | Test Repository |
|---|---|
| Test Suite | Should contain Sub Folders, Exception Handlers, Global Object files, Set Data file, Driver Scripts, Initialization & |

| | Termination scripts |
|---|---|
| Driver Script | Object Checks, Bit Map Checks, Text Check, Web Check, User defined Functions, Global test Report Folder |
| Driven Script | GUI/Bit/Text Check, External Libraries, I/O Handlers |

### 1.3 Control Points

In any given automation tool the overall control of AUT is by Object identification technique. By this unique feature the tool recognizes the Application as an medium to interrogate with the Tester supplied inputs and tests the mobility of the Business Logistics. Invoking this object identification technique the test tool does have certain control features that checks the application at various given point of time. Innumerous criteria, myriads of object handlers, plenty of predefined conditions are the features that determine the so-called object based features of the Functional Check points. Each tester has a different perspective of defining the Control points.

#### 1.3.1 If. … Else:

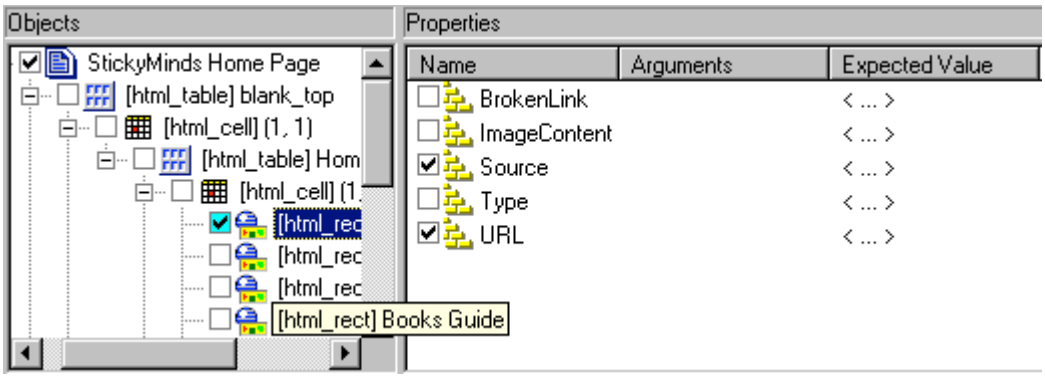
1. Before we start the "if else" construct the nature of the control point is commented along side.
   For e.g.,
   # Home Page Validation
   ```
   If (<return-code> == "0")
     print ("Successfully Launched");
   else
     print ("Operation Unsuccessful");
   ```
2. For all Data Table operation the return-code of the Open function should be handled in the "if else" construct.

#### 1.3.2 Check Points

1. Any checkpoints should not be a component of X & Y Co-ordinate dependant. In practical terms if there is a Check point that is defined on X,Y Parameters then the usability of the control point wouldn't make any sense for the application to test. The following are some of the criteria which denotes the do's and don't's of checkpoints.

| S.No | Check Point | Include | Exclude |
|---|---|---|---|
| 1 | Text Check | Capture Text, | Position of the Text, Font & Font Size, Text area, |
| 2 | Bitmap Check | only the picture | Window or Screen that holds the picture, x-y co-ordinates, |
| 3 | Web Check | URL Check, orphan page | Avoid any text validation |

2. As a case study, the WinRunner automation tool is mentioned here as examples for creating check points. Usage of OBJ_CHECK_INFO or WIN_CHECK_INFO can be avoided and inculcate the idea of creating always the GUI Check point with Multiple Property. The advantages are to identify every small object with its clause, properties and its relativity with the previous versions. This not only enables the Regression comparisons but also it gives you the flexibility of defining the GUI Checks in all Physical state of the Object.




### 1.4 Data Access

In automation Test Data becomes very critical to control, supplement and transfer in the application. In automation tools the Test Data is handled in data sheets of Excel format or a .csv file that is basically a character separated file using the Data driven technology. In most regression batch testing the Test Data is handled in data tables with proper allocation of test data in the sheets.

#### 1.4.1 Data Handlers

Test Data can be accessed by built-in data functions. Some of the common practices that would help a automation tester to use the data-tables in a proper fashion.

1. SINGLE DATA TABLE: By default every automation tool gives the data-table as an input file can be created using a tool wizard or sometimes potentially creating using a character-separated file. This wizard would help us in creating a Data sheet with its column names from the objects used in the test objects. With this concept, we can evolve a technique to load any File or manipulate the AUT by predefined set of cases.

2. Multiple Data Table: It's a common practice to use the single default data file for many test scripts. Often the usage of data tables is restricted to one file at a moment. Handling multiple data tables is not advisable and incur a lot of redundant code to handle the table manipulations. As a general practice the data file is mapped to every script. This mean every Test Script will have a unique data table for easier data access and also the data operation will become easy to maintain.

In Compuware's QARun following is the code used.

```
// Run a test script
  TestData ("CreditLogon.csv")
  Call TestFunc1
```

For e.g in Mercury Interactive's WinRunner,

```
        call_close "Test_Script1" (dTable1.xls) ;
        #
        call_close "Test_Script2" (dTable2.xls);
```

3. Data files should be initialized before starting by way of simple tool commands by transferring a standard template data table to the actual template. By this practice the need of deleting data after every run in the data table can be avoided.

   In Mercury Interactive's WinRunner the piece of code below explains the data table Initialization.

```
#/***************Data Table Initialization*****************
ddt_open(Template, DDT_MODE_READ);
ddt_open(dTable, DDT_MODE_READWRITE);
ddt_export(Template,dTable);
ddt_save(dTable);
ddt_close(dTable);
ddt_close(Template);
ddt_close_all_tables();
#/***************Data Table Initialization*****************
```

4. Dynamic loading of data from the Data base operation is the most advisable practice to be followed, but yet handling the db operations with some meticulous programming would always benefit the tester avoiding a variety of operational hazard and reducing the data access time for remote server database to the local data table.
   Some of the tips, which need to be followed in the WinRunner TSL handling when we use the db commands.
   Set the row before writing the data values in to the data-table.
   i.e., Use the following TSL Command
   public count;
   count = 1;
   ddt_set_row (dTable, count);
   Now we use the set value by row command for writing the values in it
   ddt_set_val_by_row (dTable, count, "CTS_EMP_NAME", value);
   Need less to mention here, but to avoid confusion it is better to use the same column names as found in the Data Base table. And never insert any columns before or after or in between the column names in the WinRunner data table. It is a better practice to load the data table with the data as found in the Backend database.

   Fig 1. shows the Automation Test Plan, its pre-requisites, Initial Conditions and the Test repository. This figure also gives the idea of building any Automation Test plan.

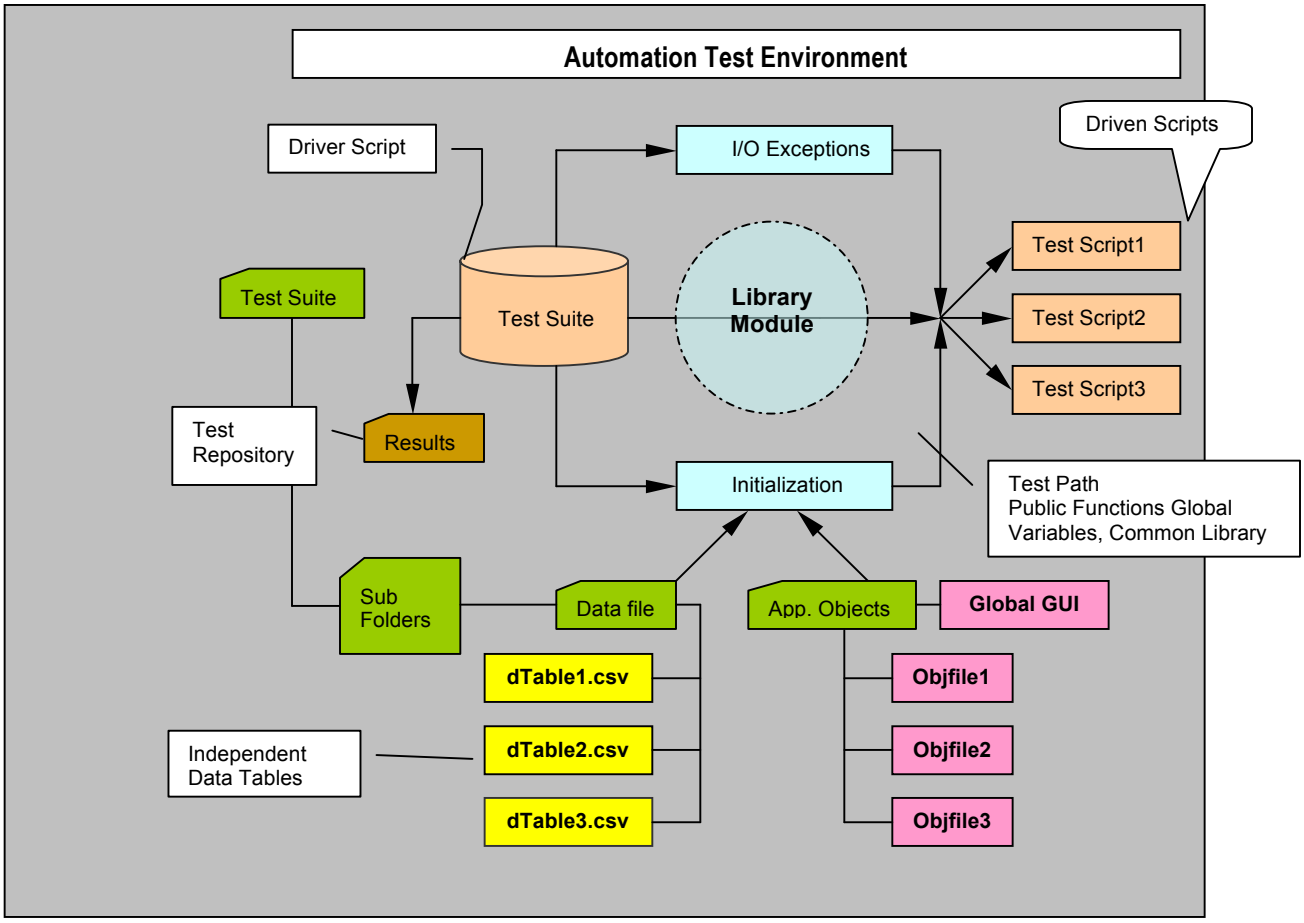



**Fig: 1**

## 2. Execution of Tests

**Best Practices and Guidelines for better Test Execution**
**-** Common Questions, What, Why and When should we practice them

### 2.1 Online Vs Batch Execution –

#### 2.1.1 Online Test Scripts

**1. Q**. How do we use Online Scripts?

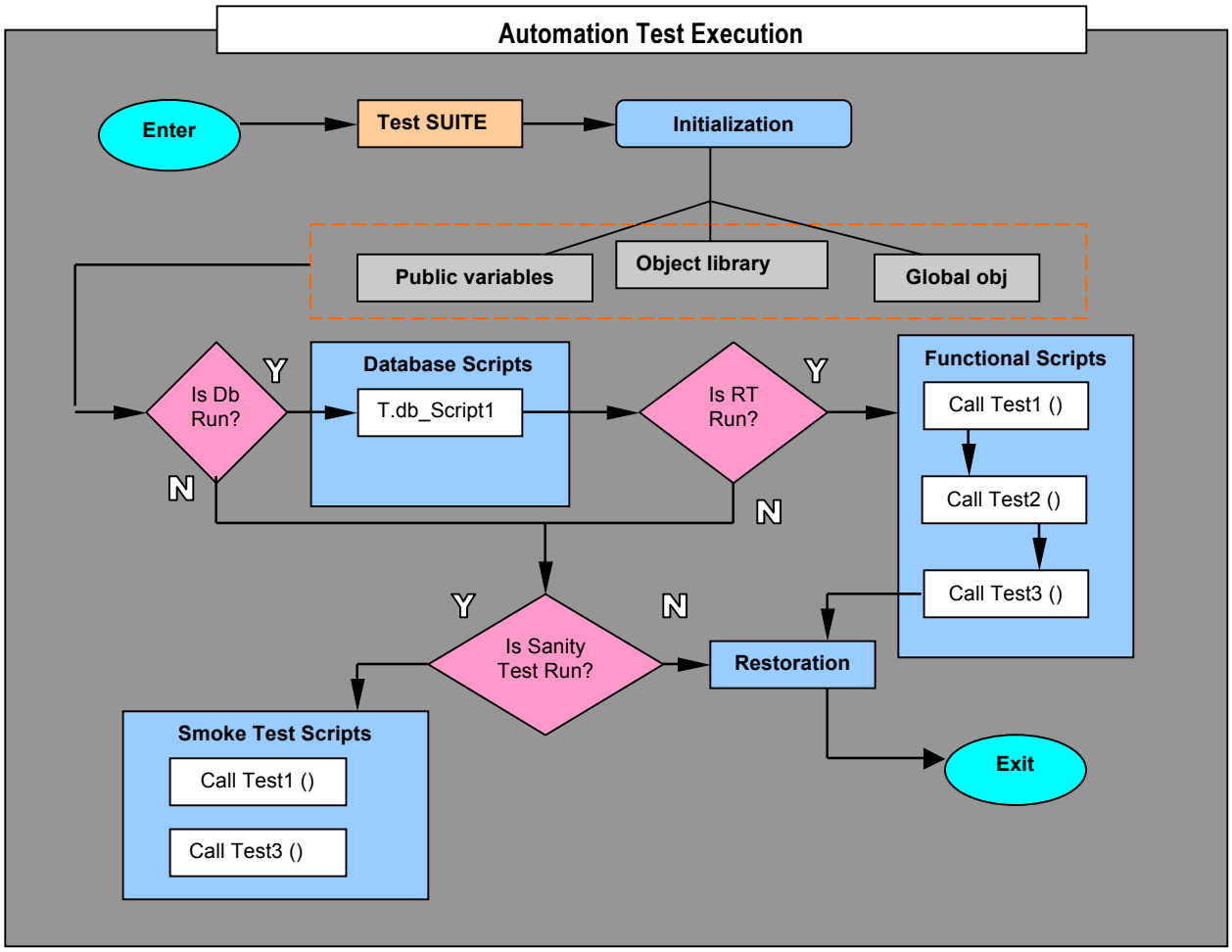
**A.** Using Dialog functions we can use the Interactive Testing accomplished.

```
In Mercury Interactive the following code is
  SSN = create_input_dialog ("Please Enter the SSN Number");
In Compuware's QARun the following code is
  Dialog "Array_A" Array_A []
  USER = Array_A["Userid"]
  Pass = Array_A["Password"]
```

#### 2.1.2 User Input

**2. Q.** Where should the input_dialog_box function exist - in the driver file or in individual script?

**A.** The input dialog function should be used within the driver files (Master driver and within each of the Type driver files)




#### 2.1.3 Test Results

**3. Q.** Is it necessary to pass results back to the driver script even if scripts are not dependent? How should the results be passed back?

**A.** No need to pass the result back to the driver script if your scripts are independent.

#### 2.1.4 Re-Runnable Tests

**4.Q.** Should setup scripts be made re-runnable? If yes then why? Also what is the best way to make them re-runnable (should it be attaching a random-number string or should it be, 'if' statements to check if data already exists)

**A.** It is best to create scripts that are re-runnable but we understand that it may not be possible for all cases for Set-Up type.

**5.Q.** Calling a driver file from within a driver file? Is this advisable?

**A.** No.

#### 2.2 Functions & Compiled Modules

##### 2.2.1 Load Library

1. Loading libraries and memory issues, i.e. if a library contains 100 functions and only one function is used then unnecessarily we are loading all the function into memory. Should we make multiple smaller libraries and load and unload libraries frequently or just have one big library and keep it loaded all throughout the execution of master driver

**Known Issue**

We will run into memory issues when loading 100 functions into memory

##### 2.2.2 Data Fetch

**2.Q.**Should we open and read from data table in driver scripts? Why or why not?

**A.** The purpose of the driver script is to setup the application and then calls each individual scripts. To open, read and close the data-file should happen at the individual test script level.

##### 2.2.3 User Defined Functions

**3. Q.** Creating user-defined libraries and functions: How to access if a script should be made a function - What are the pros and cons of making a script a function versus just using it as a script and calling it from the driver file

**A.** You have to load the function library first before you are able to make a call out to any of the functions defined in a function library. Using User-defined function is more efficient in the sense that they are compiled and loaded into memory before a function is being called and a function can be used over and again without having to recompile the function library.

##### 2.2.4 Wild Card Characters

**5.Q.** Every time there is a change in the Application Object I need to change the Object name and rerun the Test Script with a new object Name. Any suggestions on it.

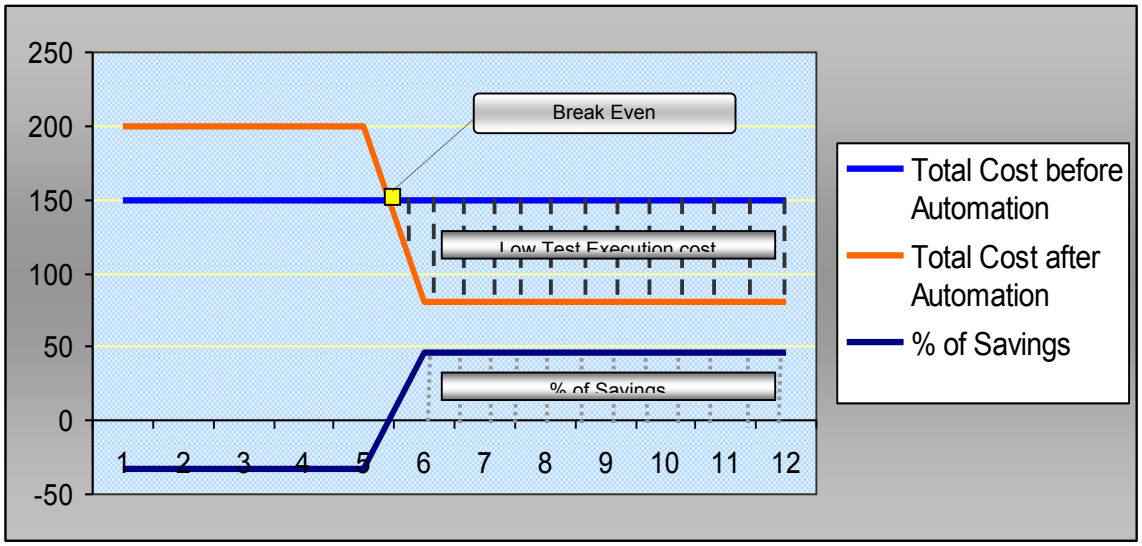
**A.** If there is a minimal change in the application Object then it is better to wild card the Object properties.

### 3. Automation ROI

Tabulated below is a matrix of Manual Vs Automated test execution effort. It gives an overall comparison of the released based regression testing effort. Calculated wholly on a basis of Release/month for regression test execution. Also the factors that influence would be

1. Regression Test Defects
2. Re-Testing effort
3. Retrospective analysis to add/delete Test Cases for change in functionality
4. Factors that influence most will be minimum period of Regression test execution
5. Symmetric data – Simulating Test Data from Functional testing test cases

| | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 | R12 | Annual Savings |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | |
| Manual Testing | 150 | 150 | 150 | 150 | 150 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | |
| Automated Testing | 50 | 50 | 50 | 50 | 50 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | |
| Total Cost before Automation | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 1800 |
| Total Cost after Automation | 200 | 200 | 200 | 200 | 200 | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 1560 |
| % of Savings | -33.33 | -33 | -33 | -33 | -33 | 47 | 47 | 47 | 47 | 47 | 47 | 47 | 13.33333333 |




**Glossary of Terms**

| | |
|---|---|
| **ROI** | **Return of Investment** |
| **GUI** | **Graphical User Interface** |
| **DB** | **Data Base** |
| **dTable** | **Test Data File or Table** |
| **AUT** | **Application Under Test** |
| **I/O** | **Input/Output** |