## Define Complexity of Design

### Write robust test scripts with a very low level of complexity.

| Value | • Decreases maintenance costs.  
|       | • Encourages reusability.  
|       | • Test scripts can be handed off and maintained by junior or new members of the team with minimal effort. |

| Task | Define a level of test design complexity based on the UFT proficiency level of your QA/scripting team. |

| Consideration & Rationale | There exists a fine balance between automated script complexity and script robustness. Automated scripts have a tendency to be cast aside as time goes on and as employees turn over. The reason for this is that many organizations write very complicated scripts that can only be maintained by the original creator of the script. Although they may be very robust, the knowledge of these scripts is not easily transferred. |
## Define Test Automation Architecture

### Monitor test automation progress and proficiency

<table>
<thead>
<tr>
<th>Value</th>
<th>Knowledge of project progress and proficiency can be easily obtained at any point during the life a project lifecycle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Define your test-automation architecture</td>
</tr>
<tr>
<td><strong>Consideration &amp; Rationale</strong></td>
<td>Orasi’s preferred architecture for test automation is QuickTest Professional for Quality Center. The Quality Center Architecture comes with many benefits, including modularity of test components, remote test execution, defect tracking, and requirements correlation with test cases, automated test scripts, defects and release cycles. At the outset it is important to know what information will be required to track and measure project progress and proficiency. A set of robust reports capable of delivering a wide variety of information on many items such as test execution status, automation percent complete, and defect trends should be defined at the outset of the testing initiative.</td>
</tr>
</tbody>
</table>
HP ALM/QC - HP UFT Integration

- Storage for automated tests
  - Accessible to the entire team
  - Optional version control
- Manage the lifecycle of automated tests
  - Track/report on progress test development and maintenance
- Schedule unattended test executions
  - Results automatically collated
  - Control flow of tests within a test set
  - Set notifications and recovery for unattended test execution
## Define Conventions

### Establish coding standards

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th>Uniformity in your code improves the readability and facilitates change management in your test automation group, leading to an improved overall test script development process.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td>Define scripting conventions</td>
</tr>
<tr>
<td><strong>Consideration &amp; Rationale</strong></td>
<td>Scripting conventions are a very important part of the development process of automated scripts. These scripting conventions should not only be implemented for one group in an enterprise organization, but all groups within an organization should adopt naming and coding conventions.</td>
</tr>
</tbody>
</table>
HP UFT – Anatomy of an Automated Test

Test Script
- Action → Action → Action

Test Datasheet

Object Repository

Function Library

Recovery Scenario

Action
- Action1
  - Home - Online Banking
    - services
      - check
      - set
      - setSecure
    - Sign In
      - click
    - Account Summary - Online Banking
      - Transfer
        - fireEvent
        - click
    - Transfer - Online Banking
      - fromAccount
      - toAccount
      - amount
      - description
      - Transfer
        - click

CheckPoint("services")
"lara"
*4f040f855e8cc38bb9f4d3de2...
29,5
"onmouseover"
"Checking 8832-839XXX"
"Saving 7195-371XXX"
DataTable("TransferAmount", d..."trying to save some money"

www.orasi.com
## Standard Action Appearance

<table>
<thead>
<tr>
<th><strong>Use standard automated test scripting templates</strong></th>
</tr>
</thead>
</table>
| **Value** | • Ease of maintenance.  
• Reusability.  
• Ease of knowledge transfer across projects. |
| **Task** | Standardize action appearance |
| **Consideration & Rationale** | As all test cases should be written using standard templates, all automated test scripts should also be developed using standard templates. UFT allows the user to develop a template for the appearance of each Action within UFT. UFT captures a user’s interaction with an application and then houses the interaction as segments of code within an Action. Each Action typically will become a building block of a script that tests an outlined business process. |
# Create Object Repository Files

## Maintain a robust test automation framework

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th>Minimizes the amount of effort required to make a change to test scripts when the application under test changes.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td>Create object repository files</td>
</tr>
<tr>
<td><strong>Consideration &amp; Rationale</strong></td>
<td>In order to build a robust test automation framework, the automation specialist must determine whether or not a shared object repository or a per-Action object repository will be used. In most cases a shared object repository is the better choice because should any object property change in the application under test, a single change to the shared object repository will propagate the change to all test scripts.</td>
</tr>
</tbody>
</table>
## Establish Test Data Sets

**Develop a robust test data framework**

<table>
<thead>
<tr>
<th>Value</th>
<th>Minimizes test data development and maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td>Establish Test Data Sets</td>
</tr>
<tr>
<td><strong>Consideration &amp; Rationale</strong></td>
<td>The automation specialist should work with the business and application experts in order to determine the data needed to test each application. Factors that affect script development include determining whether or not a dataset the application uses can be refreshed, or will the script have to reset any data that was manipulated during execution. These questions need to be answered before development of automated scripts can begin</td>
</tr>
</tbody>
</table>
# Create Function Library

## Maintain data manipulation functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Maximizes the use of reusable functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Create function library</td>
</tr>
<tr>
<td>Consideration &amp; Rationale</td>
<td>In the process of reviewing and recording a business process, the automation specialist may encounter situations where data may need to be manipulated the same way on more than one occasion. If this is the case, a function which will provide the data manipulation should be created and stored in a function library so that it can be called multiple times by different test scripts.</td>
</tr>
</tbody>
</table>
## Create Action Library

### Segment the application under test into logical business components

| Value | • Minimize maintenance effort and cost  
|       | • Increase ease of knowledge transfer |

| Task  | Create action library |

| Consideration & Rationale | The automation specialist should analyze the application under test with the application expert and break it down into small, logical business components. Each miniature business component should be scripted and provided as a building block of an entire business process. For example, logging into the application would be a business component for just about any business process. Business components should be logically stored in a Quality Center file hierarchy as Action libraries. Each one of these Actions should be labeled as a reusable Action. |
# Create Actions

<table>
<thead>
<tr>
<th><strong>Identify regular and reusable actions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td><strong>Task</strong></td>
</tr>
</tbody>
</table>
| **Consideration & Rationale** | There are two main types of Actions:  
**Regular Actions** - can be used only in the test in which they were created.  
**Reusable Actions** - can be called by other Actions, either regular or reusable.  

Action Libraries can be grouped into libraries. Once the Action library has been created, the automation specialist can call all of the different building blocks (Reusable Actions) in order to make up a full test. |
# Define Recovery Scenarios

<table>
<thead>
<tr>
<th>Identify verification routines and recovery procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td><strong>Task</strong></td>
</tr>
<tr>
<td><strong>Consideration &amp; Rationale</strong></td>
</tr>
</tbody>
</table>
Tips to Improve Runtime Performance

- From Croma Campus,
1. Always set UFT to run in Fast mode instead of Normal mode

UFT scripts can be executed in two modes Normal and Fast.

- Normal (displays execution marker) — Runs your test or component with the execution arrow to the left of the Keyword View or Expert View, marking each step or statement as it is performed. If the test contains multiple actions, the tree in the Keyword View Item column expands to display the steps, and the Expert View displays the script, of the currently running action.

- Fast — Runs your test or component without the execution arrow to the left of the Keyword View or Expert View (for tests) and does not expand the item tree or display the script of each action as it runs. This option requires fewer system resources.

We can save the time which UFT takes to display execution marker and enhance the performance of our scripts.

To switch mode please select following option (Tools->options->run).
2. Always disable smart identification.

- If UFT is unable to find an object using the learner object description it uses the smart identification mechanism for object identification. The Smart Identification mechanism takes more time to identify objects in case of failures as it checks all candidate objects and tries to find a match based on base and optional filter properties. If we turn off the Smart Identification mechanism then we can improve script performance in case of failures.

- When user select this option, the Enable Smart Identification check boxes in the Object Properties and Object Repository dialog boxes are disabled, although the settings are saved. When you clear this option, the Enable Smart Identification Check boxes return to their previous on or off setting.
Always release all objects/connections to COM objects/variables after usage.

Function example() Dim a a=5 msgbox a a=Null End Function

The Nothing keyword in VBScript is used to disassociate an object variable from any actual object. Use the Set statement to assign Nothing to an object variable. For example: Set MyObject = Nothing
Use the Dictionary Object

4. Use UFT's inbuilt dictionary object instead of arrays for storing values wherever possible.

Dim Dict1 Set Dict1 = CreateObject("Scripting.Dictionary")
dict1.Add "emp1", "gaurav" ' Add some keys and items.
Dict1.Add "emp2", "sandeep" Dict1.Add "emp3", "manish"
5. Always use WaitProperty statements instead of hardcoded wait.

A wait statement will halt the script execution for the specified time irrespective of whether the object we are waiting for has loaded or not. On the other hand Wait Property waits until the specified object property achieves the specified value or exceeds the specified timeout before continuing to the next step. Syntax - object.WaitProperty (PropertyName, PropertyValue, [TimeOut]) Use Browser(.....).Page(.....).Webbutton (....).WaitProperty (―Enabled‖, True, 1000) Instead of Wait(10) In the example above the wait statement will invariably wait for the Specified 10 seconds even if the object that we are waiting for is displayed In 5 seconds
Use Time Limit with Exist Operation

- Syntax - Object.exist([Timeout])
- If the value 0 is specified, the property returns the True or False value immediately. If a timeout value is specified, QuickTest waits until it finds the object or until the timeout is reached. If no value is specified, the Value specified in the Test Settings dialog box for the Object Synchronization Timeout is used for tests. If Browser().Page().WebButton.Exist(0) Then Browser(......).Page(......).Webbutton(....).click End If
Reset Global Sync Time

- We should be using all sync and wait statements with specified timeout values so we can set the global sync time to 1 sec.
Automate UFT Test Execution

- Launch UFT by creating a script that uses the Automation Object model. The script can be used to launch UFT with the required add-ins and UFT settings.

  Set qtApp = CreateObject("QuickTest.Application")
  directoryTestRun.Run qtApp.Quit Set qtApp = Nothing
Create Object References for Frequently Used Objects

- systemutil.Run "iexplore.exe", http://in.rediff.com/ Option1
  Browser("rediff.com").Page("rediff.com").WebEdit("login").Set i Next
  Reporter.ReportEvent micWarning, "Object Repository", "Time: 
  Option1 start = Timer() Set userName = Browser("Welcome to
  rediff.com").Page("Welcome to rediff.com").WebEdit("login") For i=0 to 150
  userName.Set i Next Reporter.ReportEvent micWarning, "Wrapper Call", "Time: 
  Option3 start = Timer() Set userName = Browser("Welcome to
  rediff.com").Page("Welcome to rediff.com").WebEdit("login").Object For i=0 to 150
  userName.value= i Next Reporter.ReportEvent micWarning, "Object Call", "Time: 
  Option1 in the example above is a normal set statement and is pretty
  slow. Option2 sets a reference of the login text box in a variable and then uses this
  variable to execute the set operation and has a slightly better performance than
  option1. Option3 uses the object property of the login webedit object. The object
  property is used to access the internal methods and properties of the object. In this a
  variable reference is set to the object directly and this is extremely fast.
The select case construct can only be used in case we are evaluating just one expression and we have different paths based on the different values for the expression. The construct is much better than the if... Elseif construct in terms of performance.
Use With Statements

- We can use with statement to minimize the trips to the repository for statements that use the same parent hierarchy, and as we are not accessing the repository for every statement we save in terms of execution time.
Connect to Excel as a Database

- The datatable does provide a lot of other features but if we only need to use the data then we can create a recordset of the data in excel, also update the data and save it also for future reference only. The recordset object has performance advantage over the UFT Datatable object. Example Set

```vbnet
cnnData = CreateObject("ADODB.Connection")
cnnData.ConnectionString = "Driver={Microsoft Excel Driver (*.xls)}; DBQ=" & DataSheetPath & ";" :cnnData.Open Set objRS = CreateObject("ADODB.Recordset")'Creating the Record set Object. With objRS .Source = "SELECT * FROM ["& SheetName & "] where TESTCASENAME<>""&""&"".ActiveCon```
Use Custom Exception Handling

- UFT provides the Recovery scenario wizard to handle exceptions. The wizard allows us to define the mechanism to create recovery scenarios that we can then associate with our tests. The mechanism however degrades the performance of the scripts in case of exceptions. The recommended way to handle exceptions is to use the On Error Resume Next, Err object to handle exceptions inside the functions and Actions. The objects are built into VBscript and a judicious use of them leads to an improvement in performance.
Use Judicious mix of Actions and Functions

- UFT provides an option to use Actions/Functions to modularize our code. The Actions have a repository and datatable attached to them. We should use a judicious mix of Actions and Functions. Actions are bulky, and create a lot of folders and objects and are also slower to execute. We can do almost everything we can do using Actions by using Functions as well. Functions are easier to create, are lightweight as they are just plain text and are stored as either .qfl or .vbs and are also better in terms of performance.
Put Declarations in a Separate File

- If we declare variables inside a reusable action then on every iteration memory (RAM) is allocated to these variables and it is not released until execution is completed.
In case of website automation a lot of application load time is taken up by downloading images. If we are not validating images then we can turn them off that will speed up the application response time by a great deal and thus the automation scripts. To disable downloading of images go to the Tools → advanced menu of Internet Explorer and uncheck the option show pictures.
The different property sets should be set up with the properties using which you can easily identify the objects on your Application under test. To check/modify the settings go to Tools → Object identification dialog in UFT.
Coding Standards

- Use Option Explicit.
- Use an ampersand (&) for concatenating strings instead of ‘+’.
- Set the objects to nothing for cleaning the memory.
- Declare only one variable in a line.
- The code should be properly indented.
- Framework code should be designed and developed with minimal possible loops and conditions for reduced complexity and enhanced maintainability.
- Modular code for increased re-usability and reduced redundancy.
- Code should be well indented with tabs. (Tab width should be 4).
- Values passed and returned to the functions should use simple variables.
- Reduce the use of global variables within the function. The scope of the variable should be decided based on the standards.

Wednesday, April 17, 2013
Start your automation work only when AUT (Application Under Test) is stable. i-e You should have completed initial round of manual testing before starting UFT automation task.

Use reusable actions wherever possible.

Don’t copy and paste checkpoints when you wish to check same thing in more than one places. Instead, create separate checkpoints. Because manipulating checkpoints in UFT is not feasible or it will be difficult.

Open browser/application after opening UFT.

Your vbscript functions should not have any hard coded object names, the name of the objects should be passed as function argument.

Use Shared Object Repository. My suggestion is don’t use DP (Descriptive Programming) which will increase your maintenance work.

Before adding any object to Object Repository, set up appropriate object identification properties (in Tools->ObjectIdentification) if you feel that default identification properties are not suitable for you application. If you do it after adding some objects to the Object Repository then you may face issues related to having multiple Test objects in Object Repository for a single actual Object.

Open UFT using Automation object model. You can do it just by writing (even you are having ‘generate script’ option in the UFT IDE itself) simple .vbs file, and then open UFT just by double clicking this .vbs file. It is very very essential if you are developing your scripts in one machine/environment and planning to run/execute it in another environment. By using Automation Object model we are making sure that all the IDE settings (e.g Object identification properties) that are changed in development environment are propagated to the execution environment also.

Always use Reference/Relative path when calling any external reusable action instead of using absolute/full path, Otherwise the script will fail when placing it in different path.

Remember to take backup of Object Repositories before merging object repositories.

Put proper inline comments and also put summary comments (description, input & output parameters, dependency, author) at beginning of the Test. In the comments remember to mention about execution/data flow.

Rename the Objects in Object Repository to have proper meaning for them. Because the default name given by adding the objects by recording or manually may not be good/meaningful.

Give attention to synchronization (i-e use of sync, waitproperty, exist). And also use regular expression wherever required. Otherwise consistency of script execution will be affected.

Use recovery scenario to handle any unexpected behaviour (e.g showing pop-up windows sometimes) of the application.

---

Best Practices in UFT Automation

By Rajamanickam Antonimuthu

Start your automation work only when AUT (Application Under Test) is **stable**. i.e. You should have completed initial round of manual testing before starting UFT automation task.

Use reusable actions wherever possible. And, Functional libraries should be used to include the vbscript functions so that maintenance of vbscript functions is easy.

Don’t copy and paste checkpoints when you wish to check same thing in more than one places. Instead, create separate checkpoints. Because manipulating checkpoints in UFT is not feasible or it will be difficult.

Open browser/application after opening UFT.

Your vbscript functions should not have any hard coded object names, the name of the objects should be passed as function argument.

Use **Shared** Object Repository. My suggestion is, don’t use DP (Descriptive Programming) which will increase your maintenance work.

Before adding any object to Object Repository, set up appropriate object identification properties (in Tools->ObjectIdentification) if you feel that default identification properties are not suitable for you application. If you do it after adding some objects to the Object Repository then you may face issues related to having multiple Test objects in Object Repository for a single actual Object.

Open UFT using **Automation object model**. You can do it just by writing *[even you are having ‘generate script’ option in the UFT IDE itself]* simple .vbs file, and then open UFT just by double clicking this .vbs file. It is very essential if you are developing your scripts in one machine/environment and planning to run/execute it in another environment. By using Automation Object model we are making sure that all the IDE settings (e.g Object identification properties) that are changed in development environment are propagated to the execution environment also.

Always use **Reference/Relative path** when calling any external reusable action instead of using absolute/full path, Otherwise the script will fail when placing it in different path.

Remember to take **backup** of Object Repositories before merging object repositories.

Put proper inline **comments** and also put summary comments (description, input & output parameters, dependency, and author) at beginning of the Test. In the comments remember to mention about execution/data flow.

**Rename the Objects** in Object Repository to have proper meaning for them. Because the default name given by adding the objects by recording or manually may not be good/meaningful.

Give attention to **synchronization** (i.e use of sync,waitproperty, exist). And also use regular expression wherever required. Otherwise consistency of script execution will be affected.

Use **recovery scenario** to handle any unexpected behaviour *(e.g showing pop-up windows sometimes)* of the application.


Posted by **UFT** on August 23rd, 2011 at 6:47 am
Thank You!