Optimizing Modular Test Automation

Presented By: David Dang
• Introduction
• Background on test automation approaches
• What is a modular test automation approach
• The advantages of a modular test automation approach
• The pitfalls of a modular test automation approach
• How to enhance the modular test automation approach
• Conclusion
Background test automation approaches

- Record/Playback
- Data-Driven
- Modular
- Keyword
- Database
What is a modular test automation approach?

A modular test automation approach decomposes the application under test into functions or modules. The functions or modules are linked together to form automated test cases.

This approach is modeled after development approaches such as functions, classes, or web services.
Core Components of Modular Test Automation

- Automated Modules
- Object Map/Repository
- Data Files
- Automated Test Cases
Modular Test Automation Example

Test Case: TZ01
Objective: Verify Time Zones are correctly set and displayed, using the Time Zone selection tab on the Date & Time Properties.
Automated Modules

1. Test Name: Login Module
2. Description: This module will invoke the Flight Application and input the user name and password
3. Author: David Dang
4. Date: 1/28/2010

5. Invoke Flight Application
6. SystemUtil.Run "C:\Program Files\HP\QuickTest Professional\samples\flight\app\flight4a.exe"

7. Login into Flight Application
8. Dialog("Login"), WinEdit("Agent Name"), Set datatable("Agent_Name", dLocalSheet)
9. Dialog("Login"), WinEdit("Password"), SetSecure datatable("Password", dLocalSheet)
10. Dialog("Login"), WinButton("OK") Click

11. Test Name: New Reservation Module
12. Description: This module will input the information needed to create a new reservation
13. Author: David Dang
14. Date: 1/28/2010

15. Set initial condition for this module, click on new order button
16. Window("Flight Reservation"), WinButton("Button_5") Click

17. Input data for new reservation
18. Window("Flight Reservation") ActiveX("MaskEdBox") Click 0.6
19. Window("Flight Reservation") ActiveX("MaskEdBox") Type DataTable("Date_of_Flight", dLocalSheet)
20. Window("Flight Reservation") WinComboBox("Fly From") Select DataTable("Fly_From", dLocalSheet)
21. Window("Flight Reservation") WinComboBox("Fly To") Select DataTable("Fly_To", dLocalSheet)
22. Window("Flight Reservation") WinButton("FLIGHT") Click
23. Window("Flight Reservation") Dialog("Flights Table") WinButton("OK") Click
24. Window("Flight Reservation") WinEdit("Name") Set DataTable("Passenger_Name", dLocalSheet)
25. Window("Flight Reservation") WinRadioButton("First") Set
26. Window("Flight Reservation") WinButton("Insert Order") Click
Object Map/Repository
### Data File

#### Data Table

<table>
<thead>
<tr>
<th>D5</th>
<th>Agent_Name</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>David</td>
<td>4b67315be601f6735511fe74b439e9fda9839c87</td>
</tr>
</tbody>
</table>

#### Data Table

<table>
<thead>
<tr>
<th>D2</th>
<th>Date_of_Flight</th>
<th>Fly_From</th>
<th>Fly_To</th>
<th>Passenger_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123010</td>
<td>London</td>
<td>Denver</td>
<td>David</td>
</tr>
</tbody>
</table>

---

**Note:** The table and diagram are placeholders for data related to flight information and agent names. The data is not actual and is used for demonstration purposes.
Automated Test Case
When to use a modular test automation approach

- Application under test (AUT) has well defined functionality
- Business objective (end-to-end) test cases that test business rules and data variations
- Over 100 targeted test cases
- Test automation engineers with programming background
- Less than 200 automated modules to cover AUT
- QA team has control over test environment and test data refresh
When not to use modular test automation approach

• AUT is still under new development
• Test cases that verify:
  • Single functionality
  • Application layout or objects
  • Back-end functionality (file feed or DB verification)
• Requires over 200 automated modules
• Test automation engineers with limited programming background
The advantage of a modular test automation approach

- Mirrors most development processes
  - Development
    - Functionality is separated using functions, procedures, web services, classes, etc…
    - Functionality is called to create application
    - Data abstraction is used to pass data in and out of the application
    - Coding processes and standards are established
The advantage of a modular test automation approach

- Mirrors most development processes (cont’d)
  - Test Automation
    - Functionality is separated into modules
    - Modules are linked together to form automated test cases
    - Data abstraction is used to pass input test data
    - Scripting processes and standards are established
The advantage of a modular test automation approach

- Encourages Reusability
  - Automation modules can be used by multiple automated test cases
  - Functionality is not duplicated in multiple automated test cases
  - Test objects are shared across automated test cases
  - Data are separated from the automated modules
  - Separation between automation developer and automation user
The advantage of a modular test automation approach

- **Reduced Maintenance**
  - Single point of maintenance for:
    - Modules
    - Objects
    - Data
  - Most changes are made in the module or object map/repository and not in the automated test cases
  - Automated test cases act as “dummy” tests
  - Automation developers are responsible for modules and objects change
  - Business testers are responsible for data changes
The pitfalls of a modular test automation approach

- Data Constraints
  - Passing data between modules
  - Date sensitivity
  - Unique data
Data constraint

- Passing data between modules

Test Name: Open Flight Reservation
Description: This module will use the order number to open an existing flight reservation.
Author: David Dang
Date: 2/1/2010

Input order number to open an existing flight reservation
Window("Flight Reservation").WinMenu("Menu") Select "File: Open Order..."
Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Order No.").Set "ON"
Window("Flight Reservation").Dialog("Open Order").WinEdit("Edit").Set DataTable("Order_Number", dtLocalSheet)
Window("Flight Reservation").Dialog("Open Order").WinButton("OK").Click

Need order number from New Reservation Module

Data Table

<table>
<thead>
<tr>
<th>A1</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Order_Number</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data constraint

- Date Sensitivity
Data constraint

• Unique Data
The pitfalls of a modular test automation approach

- Not Flexible
  - Accommodate multiple functionalities within a single module
  - Application branching
  - Data targeted functionality
Not flexible (cont’d)

1. **Test Name: Open Order by Order Number**
2. **Description:** This module will use the order number to open an existing flight reservation.
3. **Author:** David Dang
4. **Date:** 2/1/2010
5. **Input order number to open an existing flight reservation**
6. Window("Flight Reservation").WinMenu("Menu").Select("File\:Open Order...")
7. Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Order No").Set "ON"
8. Window("Flight Reservation").Dialog("Open Order").WinEdit("Edit") . Set Data Table("Order\_Number", dtLocalSheet)
9. Window("Flight Reservation").Dialog("Open Order").WinButton("OK").Click

1. **Test Name: Open Order by Name**
2. **Description:** This module will use the name to open an existing flight reservation.
3. **Author:** David Dang
4. **Date:** 2/1/2010
5. **Input name to open an existing flight reservation**
6. Window("Flight Reservation").WinMenu("Menu").Select("File\:Open Order...")
7. Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Customer\_Name").Set "ON"
8. Window("Flight Reservation").Dialog("Open Order").WinEdit("Edit\_2") . Set Data Table("Customer\_Name", dtLocalSheet)
9. Window("Flight Reservation").Dialog("Open Order").WinButton("OK").Click
10. Window("Flight Reservation").Dialog("Open Order").Dialog("Search\_Results").WinButton("OK").Click

1. **Test Name: Open Order by Name and Date**
2. **Description:** This module will use the name to open an existing flight reservation.
3. **Author:** David Dang
4. **Date:** 2/1/2010
5. **Input name and date to open an existing flight reservation**
6. Window("Flight Reservation").WinMenu("Menu").Select("File\:Open Order...")
7. Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Customer\_Name").Set "ON"
8. Window("Flight Reservation").Dialog("Open Order").WinEdit("Edit\_2") . Set Data Table("Customer\_Name", dtLocalSheet)
9. Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Flight\_Date").Set "ON"
10. Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Flight\_Date").Set "ON"
11. Window("Flight Reservation").Dialog("Open Order").ActiveX("Mask\_EdBox") . Type Data Table("Date\_of\_Flight", dtLocalSheet)
12. Window("Flight Reservation").Dialog("Open Order").WinButton("OK").Click
13. Window("Flight Reservation").Dialog("Open Order").Dialog("Search\_Results").WinButton("OK").Click
The pitfalls of a modular test automation approach

- Not Easy to Logically Break Down Modules
  - Most applications have complex functionality
  - Dynamic sections within application that can change from day-to-day
  - Sections that get repeated on every window or page
Not easy to logically break down modules (cont’d)
The pitfalls of a modular test automation approach

- Verification Overlapping
  - Verify everything within a module
  - Verify nothing within a module
  - Create separate module just for verification
Verification Overlapping

```
10: Window("Flight Reservation").ActiveX("MaskEdBox").Click 0.6
11: Window("Flight Reservation").ActiveX("MaskEdBox").Type Data Table("Date_of_Flight", dtLocalSheet)
12: 
13: Verify Date of Flight
14: vDateOfFlight = Window("Flight Reservation").ActiveX("MaskEdBox").GetROProperty("Text")
15: If vDateOfFlight = Data Table("Date_of_Flight", dtLocalSheet) Then
16:   Reporter.ReportEventabicPass, "Verify Date of Flight", "The displayed date of flight is correct. The value is " & vDateOfFlight & ""
17: Else
18:   Reporter.ReportEventabicFail, "Verify Date of Flight", "The displayed date of flight is incorrect. The expected value is " & Data Table("Date_of_Flight", dtLocalSheet)& " and the actual value is " & vDateOfFlight & ""
19: End If
20: 
21: Window("Flight Reservation").WinComboBox("Fly From").Select Data Table("Fly_From", dtLocalSheet)
22: 
23: Verify fly from
24: FlyFrom = Window("Flight Reservation").WinComboBox("Fly From").GetROProperty("Value")
25: If FlyFrom = Data Table("Fly_From", dtLocalSheet) Then
26:   Reporter.ReportEventabicPass, "Verify Fly From", "The displayed fly from is correct. The value is " & FlyFrom & ""
27: Else
28:   Reporter.ReportEventabicFail, "Verify Fly From", "The displayed fly from is incorrect. The expected value is " & Data Table("Fly_From", dtLocalSheet)& " and the actual value is " & FlyFrom & ""
29: End If
```
Solutions:

- Develop guidelines to break up modules.
- Develop data strategy to handle dynamic data.
- Use “keywords” to enhance the module.
- Use flags for logical branching.
Develop guidelines to break up modules

- Identify rules to break up modules
  - Each window/page
  - Repeated components within a window/page
  - Grouping of functionality within a window/page
  - Section within a window/page
  - Combining small windows/pages into one module

- Establish modules inventory
- Assign priority to each module
- Understand the effect of data variation on the module
Develop guidelines to break up modules (cont’d)

How would you break up this page into modules?
Develop data strategy to handle dynamic data

- Identify data constraints
  - Date, e.g., application only accepts future date
  - Unique data, e.g., SSN, Account number
  - Existing data from database
- Develop algorithm to generate data
- Incorporate method into data file
Develop data strategy to handle dynamic data (cont’d)

- Date Algorithm

```plaintext
1: 'Check to see if the value entered is a correct date
2: vCheckDate = IsDate(DataTable("Date_of_Flight", dtLocalSheet))
3:
4: If vCheckDate Then
5:   'Set date into date of flight
6:   Window("Flight Reservation").ActiveX("MaskEdBox").Type = DateTable("Date_of_Flight", dtLocalSheet)
7: Else
8:   'If not, we need to forward the date from current date
9:   vCurrentDate = Date
10:   vNewDate = DateAdd(d, DataTable("Date_of_Flight", dtLocalSheet), vCurrentDate)
11:   Window("Flight Reservation").ActiveX("MaskEdBox").Type = vNewDate
12: End If
```

<table>
<thead>
<tr>
<th>Date_of_Flight</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/30/2010</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date_of_Flight</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Use “keyword” to enhance the module

- Develop Action keywords to establish intent
  - Input
  - Verify
  - Query
- Use conditional statements to drive scripting
- Incorporate method into data file
Use “keyword” to enhance the module (cont’d)

- Example of using keyword to enhance the module

```plaintext
   Check Action to determine script path
1. If DataTable("Action",dLocalSheet) = "Input" Then
2.   If DataTable("UserName",dLocalSheet) <> "" Then
3.      Dialog("Login"), WinEdit("Agent Name:"), SetDataTable("UserName",dLocalSheet)
4.   End If
5. End If
6. Else
7.   If DataTable("Action",dLocalSheet) = "Verify" Then
8.      vUserName = Dialog("Login"), WinEdit("Agent Name:"), GetRProperty("Value")
9.      If vUserName = DataTable("UserName") Then
10.         Reporter.ReportEvent midPass,"Verify User Name","The displayed username is correct. The value is "$vUserName""
11.     Else
12.         Reporter.ReportEvent midFail,"Verify User Name","The displayed user name is incorrect. The expected value is "&Data Table("UserName")&" and the actual value is "$vUserName"
13.     End If
14. End If
```

Keyword View / Expert View

Data Table

<table>
<thead>
<tr>
<th></th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Verify</td>
</tr>
<tr>
<td>2</td>
<td>Input</td>
</tr>
<tr>
<td>3</td>
<td>Input</td>
</tr>
<tr>
<td>4</td>
<td>Input</td>
</tr>
</tbody>
</table>

A1 | Action | UserName | Password | OK_Button | E | F | G | H | I | J | K | L | M | N | O | P | Q |
1: | Verify |         |         |           |   |   |   |   |   |   |   |   |   |   |   |   |
2: | Input  | David   | mercury |           |   |   |   |   |   |   |   |   |   |   |   |   |
3: | Input  | David   |         |           |   |   |   |   |   |   |   |   |   |   |   |   |
4: | Input  | David   | *****   |           |   |   |   |   |   |   |   |   |   |   |   |   |

Global A, Action1
Use flag for logical branching

- Determine area within the application that needs logical branching
  - Payment Type, e.g., Credit Card, PayPal, Bill me later
  - New objects are activated based on data input
  - Radio group selection
- Use conditional statement to drive scripting
- Incorporate method into data file
Use flag for logical branching (cont’d)

- Example of using flag to handle payment type

```
' Check Payment Type to determine objects that will need input
If DataTable('Credit_Card_Flag', dLocalSheet) = "Yes" Then
    Window('CheckOut').WinCheckBox('Credit Card').Set "ON"
End If

Select Case DataTable('Credit Card Type')
    Case 'Master Card'
        Window('CheckOut').WinCheckBox('Master Card').Set "ON"
    Case 'Visa'
        Window('CheckOut').WinCheckBox('Visa').Set "ON"
    Case 'American Express'
        Window('CheckOut').WinCheckBox('American Express').Set "ON"
End Select

Else
    If DataTable('PayPal_Flag', dLocalSheet) = "Yes" Then
        Window('CheckOut').WinCheckBox('PayPal').Set "ON"
    End If
End If
```
Conclusion

The benefits of using a modular test automation approach are enormous. However, it requires:

• Treating test automation as a development project
• Advanced planning from test automation team
• Understanding of upfront cost vs. long-term return
• Dedicated resources