This presentation is for everybody who makes decisions regarding GUI test automation:

- Managers of functional test teams
- Managers of GUI test automation teams
- GUI test automation specialists
Agenda

1. Introduction of mainstream GUI testing tools
2. Major test automation approaches/frameworks
3. Test automation process
4. Managing GUI test automation

Backup slides:
1. Alternative GUI testing tools/frameworks
2. Offshore outsourcing of GUI test automation
“Automated testing” = testing by a computer; In this case the design of a test procedure and identification of expected results are being performed by a computer.

- “Dumb Monkey” for Rational Visual Test

This presentation is about: “Regression GUI test automation”

“Regression GUI test automation” = automation of the execution of a manual functional test which was developed and documented by a human

- Functional test scripts have to be created before the start of GUI test automation.
1 – Introduction of GUI testing tools
The most efficient GUI test automation tools are the ones from major vendors:

- Mercury WinRunner and QuickTest Pro,
- Segue Silk,
- Compuware QARun,
- IBM Rational Robot,
- etc.

These tools are used by independent testing teams to automate functional test cases.

The focus of this presentation is on the efficient implementation of these tools.
Myth #1 – Commercial GUI testing tools are expensive.

- Under the influence of this myth some people:
  - try to develop their own testing tools to save money,
  - use scripting languages like Perl and Ruby to automate functional test cases and
  - do not consider test automation at all.
- GUI testing tools are indeed much more expensive than such development tools as Microsoft Visual Studio or Borland JBuilder,
- But are they really expensive?

High prices of GUI testing tools are just a myth!
Reality – GUI testing tools are cheap

- Per seat license for one of the most expensive GUI testing tools is about $8,000.00.
- This tool will be used for two years (and then thrown away or put on a shelf). Cost of this tool is $4,000.00 per year.
- Cost of test automation person with overhead is $80,000.00 per year.
- The cost of a GUI testing tool is just 5% of the cost of the person who will use it.
- The increase of productivity of a person who uses these tools is more than 5% compared to a person who uses other alternatives.
Record and Replay Test Automation Approach:

- Tester executes manual test script while GUI testing tool records all his actions:
  - Mouse clicks
  - Key presses
  - Pauses/delays

- Then GUI testing tool replays the script exactly as it was recorded, including:
  - Timing of script execution
  - Data entered into different fields
GUI test automation tool – an example
GUI test automation approaches

2 – GUI test automation frameworks
Record and Replay test automation – Pros

- The record and replay approach is relatively simple and fast
- The record and replay approach might be useful when you change something beyond GUI, e.g.:
  - Back-end configuration parameters,
  - Back end OS,
  - Version of back-end product
  - etc.,
Record and Replay test automation – example

- Example of successful implementation:
  - Verification that online bill looks exactly the same after a new version of a back end Bill Presentment system is implemented.

- Vendors were extremely sensitive about the exact appearance of their bills:
  - Corporate logos should be valid up to a pixel,
  - The layout of an electronic bill should be pretty close to the paper version; it should be exactly the same regardless of all changes to the back end code.

- The same user IDs, account numbers and bill amounts were used for every test run.
Record and Replay test automation – Cons

- Only one functional test case can be automated by one automated test script.
- What about changes in:
  - GUI
  - Target URL,
  - User ID/Password
  - Business data (e.g. price, order number, invoice number, posting date, etc.)?
- When anything changes (new version of a product, new build, bug fixes, patches, test data, etc.) – the script has to be rerecorded!
- **Poor maintainability**
  Very limited usefulness
Record and Replay test automation – conclusion

- The success story I described above (verification of online bills) might be only an exception.
- Applicability/utility of “record and replay” approach is very limited.
- Typically I recommend it only for training purposes.
Let’s assume that you have 27 scripts. Each script starts with the same statement:

```java
web_browser_invoke(IE, http://confut.cell.ont.ca);
```

You need to execute these scripts against different environments, e.g. QA, Mirror, Staging.

What would you do with a “classical” record and replay approach?

You develop a script for every target URL, e.g.:

- Verify_Access_QA,
- Verify_Access_Staging,
- Verify_Access_Mirror

The only difference between them – the initial URL!

**We have uncontrolled multiplication of test scripts!**
Test automation approaches / frameworks: Functional Decomposition – an example

- We start with the recorded script
- Then we introduce **functions**
- We remove this statement from all scripts:
  ```javascript
  web_browser_invoke(IE,http://confut.cell.ont.ca);
  ```
- And substitute it with a function:
  ```javascript
  UT_Site_Open();
  ```
- This function is described only in one place.
- When we need to execute these 27 scripts against another environment all we have to do is change the code in just one place!
Main test script:

```
load("Lib_UT_Functions");
UT_Site_Open( );
UT_Select_Language("English");
UT_Login("a327012", "12345");
```

Function:

```
function UT_Site_Open( )
{
    #Opens a new UT site in a new browser window:
    web_browser_invoke(IE, http://confut.cell.ont.ca);
    return 0;
}
```
Functional Decomposition – pros

- We can use a “Functional decomposition” approach when:
  - Different test scripts include identical actions (groups of test steps) e.g. “Login”.
- We substitute these repetitive groups in all scripts with functions.
- When script is broken due to any change of GUI or business logic we have to modify script code only in one place.

**Pros:** High Maintainability

- The functional decomposition method could be adequate when we have only a few parameters which can have only a few different values (e.g. target URL, user ID, browser, etc.).
Test automation approaches / frameworks: Data Driven Approach

- We start with a recorded script
- Then we substitute the recorded values with those retrieved from a data file
- One functional test case corresponds to a record in the data file
- To execute the same script with another set of test data we have to use a different record in the data file.
Test automation approaches / frameworks: Data Driven Approach – sample code

```plaintext
vTestEnvXls = "C:\Test\UTTestEnvironment.xls"

# Retrieve environment variables from a file:
ddt_open(vTestEnvXls);
vURL = ddt_val(vTestEnvXls, "URL");
vBrowser = ddt_val(vTestEnvXls, "Browser");
vUserID = ddt_val(vTestEnvXls, "UserID");
vPassword = ddt_val(vTestEnvXls, "Password");
```

<table>
<thead>
<tr>
<th>URL</th>
<th>Browser</th>
<th>UserID</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://confut.cell.ont.ca">http://confut.cell.ont.ca</a></td>
<td>IE</td>
<td>a327842</td>
<td>123</td>
</tr>
<tr>
<td>...............</td>
<td>.....</td>
<td>......</td>
<td>.....</td>
</tr>
</tbody>
</table>
Test automation approaches / frameworks: Data Driven Approach – an example

- Service address validation:

- Some “location types” were generating errors
Test automation approaches / frameworks:
Data Driven Approach – an example

- 44 location types
- Add-hoc test automation
- Script read all address fields from a data file
- One script covered 44 functional test cases
- Script was executed against several builds until this defect was completely fixed.
- This script was also executed several times for regression testing.
Data Driven Approach – conclusion

- We can use a “Data Driven” approach when:
  - The workflow is identical and
  - There are several different sets of test data for the same test procedure (workflow)

- Benefits of a “Data Driven” approach:
  - 1 script automates several test cases and
  - Test data can be prepared/verified by a functional tester or a Business Analyst.
    (everybody can use EXCEL without a training!)

- Pros:
  - **High Efficiency**
Combination of the functional decomposition and data driven approaches

- This is a combination of two previous approaches:
  - Script is broken into functions and
  - Test data are being retrieved from a data file

- This is the current industry standard
- Most successful GUI test automation teams use this approach

- Pros:
  - High Maintainability
  - High Efficiency
3 – Test Automation Process
First rule of GUI test automation

- Probably you have already heard many times the first rule of GUI test automation:

  **It makes sense to automate a test case when it will be executed about 10 times (against 10 builds/releases of an Application).**

- This is a simplified rule.
- It’s a good starting point for test planning.
- Software development shop with daily/weekly builds vs. customized implementation of a commercial product.
“Good enough” test automation scripts

- We can not justify the same quality requirements for a GUI test automation script (which will be executed only 10 times) as for an application which sells in the millions of copies
- We can tolerate more bugs in test automation scripts than in mass market software products
- The failure of a “good enough” test automation script doesn’t necessarily mean that a defect of an Application Under Test is discovered
- Often a script fails because of a defect in the test script itself or in a test data set
De facto standard of test case documentation

Sample functional test script:

1. Click on “User ID” field
2. Type in “a327012”
3. Click on “Password” field
4. Type in “12345”
   – 5 stars “*****” should be displayed in the “Password” field
5. Click on “Login” button
   – main application screen should be displayed.

Test procedure mixed with test data!
Current status of test documentation may lead to inefficient test automation

- There is a de facto standard of test documentation which incorporates a test procedure and test data in one so-called “test case”:
  - One test case
  - One test procedure
  - One test data set
  - One test condition

- Such functional test documentation may lead to inefficient test automation:
  - One automated test script
  - One functional test case

- It’s not a fault of the “IEEE 829-1983 Standard for Software Test Documentation”.
More efficient test automation

- It is more efficient to cover several test conditions with the same automated test script:

  One automated test script

  Several sets of test data (test conditions)

- To use this approach, we need the functional test documentation in the following format:

  One test procedure

  Several sets of test data
Efficient test automation starts with efficient test documentation

To ensure efficient GUI test automation you have two options:

1. Convince your functional test team to use a better approach to test documentation or
2. Convert the available set of functional test cases (yourself) into the following format:

One test procedure

Several sets of test data
Commercial test management tools and their integration with GUI test automation

- **Pros:**
  - Provide better “control” and an audit trail for test execution, coverage, etc.

- **Cons:**
  - Less portable and less maintainable scripts.
  - A lot of network traffic.
  - Typically these tools tend to produce very granular scripts like:
    - Click on a field “…”
    - Type in “…”
    - Click on “Save” button
  - Hard to integrate with data driven framework.
  - A type of test documentation that leads to inefficient test automation!
Myth #2 – The Test Automation Process is simple

- A tools sales person will try to convince you that the following is a real GUI test automation process:
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Report defects

- This process looks pretty simple and straightforward.

This simple process is just a myth!
Test Automation Process – reality

- Real test automation process:
  - Choose a test automation approach and develop a test automation framework
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Report defects
Test Automation Process – reality

- Real test automation process:
  - Choose a test automation approach and develop a test automation framework
  - Design test automation scripts (efficient, maintainable, portable, etc.)
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Report defects
Test Automation Process – reality

- Real test automation process:
  - Choose a test automation approach and develop a test automation framework
  - Design test automation scripts (efficient, maintainable, portable, etc.)
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Analyze causes of scripts’ failures
  - Report defects
Test Automation Process – reality

- Real test automation process:
  - Choose a test automation approach and develop a test automation framework
  - Design test automation scripts (efficient, maintainable, portable, etc.)
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Analyze causes of scripts’ failures
  - Report defects
  - Repair failed scripts
Myth #3 – GUI testing tools are simple - Every functional tester can use them

- This myth is promoted by the tool sales people.
- This myth assumes the “click, click, click”, record and replay approach.
- Under the influence of this myth a test manager proudly can report:
  - “All our testers are using GUI test automation tools”.
- Indeed:
  - You do not need a developer’s skills to develop a simple test automated script with a “click, click, click”, record and replay approach.

But, this simplicity is just a myth!
Real Test Automation Process

- Real test automation process:
  - Choose a test automation approach and develop a test automation framework
  - Design test automation scripts (efficient, maintainable, portable, etc.)
  - Record a script.
  - Enhance a script (add functions and/or data driving)
  - Execute scripts
  - Analyze causes of scripts’ failures
  - Report defects
  - Repair failed scripts
Reality – a programming background is required to use GUI testing tools

- The efficient implementation of GUI testing tools typically requires functional decomposition + data driving approach.
- You need to have some kind of a programming background to implement this approach.
- GUI test automation scripts are not as complex as a classical C/C++ or Java back-end development.
- Definitely, an experienced software developer or a Computer Science university graduate would make a good test automation specialist.
- At the same time a science or engineering education would be more than adequate for GUI test automation.
Cooperation with functional testers

- The most important success factor is good cooperation with functional testers.
- Typically, test cases are documented with the assumption of enough knowledge of the Application Under Test.
- Test automation folks need a lot of help from functional testers to understand functional test documentation.
- When functional testers understand that the goal of GUI test automation is to help them test, and when they see real results of test automation they become interested in the success of test automation and are willing to provide enough support.
Cooperation with functional testers – an example

Functional testers:
- “Here is a list of functional test cases I am bored to death with executing. Can you automate these test cases?”

Test automation specialist:
1. “I can automate these test cases easily.”
2. “These test cases would require several times more effort from me to automate.”
3. “These test cases are rather complex and time consuming to automate. I doubt I will be able to do this in the allotted time.”
4. “Look! In addition I can also automate some more test cases. It will take very little of my time and will be almost free for you!”

The role of a manager is to make the final selection after considering all trade-offs.
Managing GUI Test Automation

4 – Managing GUI Test Automation
For GUI test automation, we must define:

- The version of the Application Under Test to develop test scripts with
- The target version of the Application Under Test for script execution / maintenance
- The test environment to develop/execute automated test scripts
- A set of regression test cases to automate
- Test data (User IDs, accounts, invoices, etc.)
Managing expectations
Reality vs. ‘Click, click, click’ approach

- Higher managers typically believe in the simplistic “Click, click, click” approach that does not include some phases of the real test automation process.
- You have to educate them and explain real phases and real tasks that are being implemented.
Phases of a GUI test automation project

- At the beginning of a GUI test automation project:
  - Tool selection / Proof of Concept (Integrated vs. standalone? Can this tool drive your GUI?)
  - Development of a basic framework
- For each version/release:
  - Design of test automation scripts (efficient, maintainable, portable, etc.)
  - Development (and debugging) of automated test scripts
- For each build:
  - Execution of automated scripts / analysis of results / reporting of anomalies/defects
  - Maintenance of automated scripts - fixing of defects in scripts or test data.
- Assign all phases of a test automation process to the same person / team!
Test environment and test data

- Pay attention to the test environment and test data during the whole process from development of test scripts to execution against consecutive versions/builds of an Application to be tested.
- In a simplified case you will be able to use the same test environment and the same set of test data for the whole test development and test execution process.
- Otherwise you have to understand how these scripts will be executed against another test environment when:
  - The old set of User IDs/passwords is no longer available,
  - Old business data (account numbers, invoice numbers, etc.) are no longer available.
Test automation metrics – script development

- During the development of automated test scripts use *The number of functional test cases which were automated* as a metric. This would lead to the implementation of efficient automated scripts.

- Do not use the number of developed *test automation scripts* as a criteria of progress. This would lead to development of zillions of small simple inefficient automated test scripts.
Let’s assume that you have 27 scripts. You need to execute them against different environments (e.g. QA, Mirror, Staging) using different browser and user IDs.

What would you do with a “classical” record and replay approach?

You develop a script for every combination of a target URL, e.g.:

- Verify_Access_QA,
- Verify_Access_Staging,
- Verify_Access_Mirror

The only difference between scripts – initial URL!
Test automation metrics – script execution

- During a phase of execution of automated test scripts use the following metric:
- **The duration of execution of all test scripts since the moment new build become available until all results are analysed and anomalies/defects are reported.**
- An example:
  - Build became available at 2:00 AM.
  - Testers started execution of automated test scripts at 9:00 AM.
  - Test execution was completed at 12:00 AM.
  - Results were analyzed and anomalies/defects were reported by 2:00 PM.
  - In this example test execution took 12 hours from 2:00 AM when new build became available until 2:00 PM when all discovered anomalies were reported.
An ideal case of GUI test automation

- When functional testers arrive at 9:00 AM they should have anomalies discovered by automated scripts already reported to them!
- How we can reach this goal:
  - Start the execution of a set of automated test cases immediately after a build process was successfully finished, Application was deployed and restarted. This could be done by:
    - A build manager starting execution of automated test cases manually.
    - Integration of an automated build process and test execution.
  - Test automation team can start analysis of results of test execution one or two hours before the start of the work day of a functional test team.
  - Use a set of additional workstations and test tool licenses specifically for test execution.
  - Each test automation specialist should have at least two workstations and two GUI testing tool licenses.
GUI Test Automation - Summary

- GUI test automation tools and frameworks are evolving
- The following is the current industry standard for GUI test automation, which ensures more efficient test automation and a better ROI:
  - Mainstream GUI test automation tools (Mercury WinRunner and QuickTest Pro, Segue Silk, Compuware QARun, Rational Robot, etc.)
  - A test automation framework which uses a combination of “Functional decomposition” and “Data driving” approaches.
- Pros:
  - **High Maintainability**
  - **High Efficiency**
Happy Test Automation!

Happy Test Automation!
Happy Test Automation – Q & A

Happy Test Automation!

Questions?
Appendices
5 – Alternative tools/frameworks
Other alternatives

- Visual Test
- Perl, Ruby, etc.
- Keyword driven framework
- FIT, FITnesse, etc.
Other alternatives - IBM Rational Visual Test

- No recording capabilities
- Everything has to be coded manually
- Sample functional script:
  Start Notepad
  Display the Open dialog box,
  Enter a file name, and
  Open that file into the Notepad application

- Visual Test code:
  ```plaintext
  run "notepad.exe",nowait
  wMenuSelect("File\Open")
  wEditSetText("File name:","bootlog.txt")
  wButtonClick("Open")
  END
  ```

**It’s a museum relic!**
Other alternatives – Perl, Ruby, etc.

- Recording capability is missing
- Everything should be coded manually

- Sample Perl code (with Win32::GuiTest module):
  - Key pressing:
    ```perl
    SendKeys("O");
    SendKeys("{RIGHT}");
    ```
  - Finding text in an Application:
    ```perl
    $text = GetWindowText($window);
    $text = WMGetText($window);
    ```

This approach can be used for small scale automation by a developer
Keyword driven frameworks

First generation:
- Action Words Driven Testing by Hans Buwalda
- Keyword Driven Test Automation by Carl J. Nagle
- Totally Data-Driven Automated Testing by Keith Zambelich

Second generation:
- Worksoft's Certify
- Mercury Business Process Testing

Third generation:
- The future of the keyword driven framework
Keywords driven framework
1st generation

- An EXCEL file represents the script:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>ActionName</strong></td>
<td><strong>ActionDesc</strong></td>
<td><strong>ParaName</strong></td>
<td><strong>ParaValue</strong></td>
</tr>
<tr>
<td>2</td>
<td>LaunchBrowser</td>
<td>Step 1 - Open Home Page</td>
<td>AppFileName</td>
<td>iexplorer.exe</td>
</tr>
<tr>
<td>3</td>
<td>SelectListItem</td>
<td>Step 2 - Select Intranet Application</td>
<td>SysTreeView32</td>
<td>Store</td>
</tr>
<tr>
<td>4</td>
<td>GoPage</td>
<td>Step 3 - Click 'Note' link</td>
<td>WebLink</td>
<td>Note</td>
</tr>
</tbody>
</table>

- Script is broken into functions. Every “ActionName” has a corresponding function.
- So called “Driver script” reads this table and invokes corresponding functions to execute different “Actions”.
- You can’t use the inefficient “Record and replay” approach with a keyword driven framework.
- Pros: **High Maintainability** (due to enforced functional decomposition).
Keyword driven framework
1st generation – data driving

- You can read data from an external data file. (separate script files and test data files):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ActionName</td>
<td>ActionDesc</td>
<td>ParaName</td>
<td>ParaValue</td>
</tr>
<tr>
<td>2 LaunchBrowser</td>
<td>Step 1 - Open Home Page</td>
<td>AppFileName</td>
<td>&lt;data in the DataSheet&gt;</td>
</tr>
<tr>
<td>3 SelectListItem</td>
<td>Step 2 - Select Intranet Application</td>
<td>SysTreeView32</td>
<td>&lt;data in the DataSheet&gt;</td>
</tr>
<tr>
<td>4 GoPage</td>
<td>Step 3 - Click 'Note' link</td>
<td>WebLink</td>
<td>&lt;data in the DataSheet&gt;</td>
</tr>
</tbody>
</table>

- **Pros: High efficiency**
  (one script can execute several sets of test data)

- With keyword driven framework you use the same principles of functional decomposition (modularity) and separation of a script and data (data driving) to reach high maintainability and high efficiency.
Keyword driven approach (Carl J. Nagle)

- A complex hierarchy of "Driver scripts" is required to interpret and execute these test scripts.
- Control of execution of a set of test scripts.
Keyword driven framework
1st generation – Business Analysts

- Premise of a keyword driven framework:
  - A Business Analyst or a Functional Tester can be used for test automation (they do not need to master a complex GUI test tool).

- Reality:
  - Good Business Analysts and Functional Testers are typically in short supply.
  - Their main goal is to test the Application (develop new test cases)
  - Using them for test automation is a waste of their time.
Keyword-driven frameworks presentations

- I read many papers on these test automation tools/frameworks and attended several presentations.
- Every time there was a comparison of a “new” wonderful sexy framework against a straw man of the infamous “record and replay” approach!
- The conclusion was that this “new” framework provided undisputed benefits compared to the “classical” test automation approach.
- At the same time these people were not enthusiastic to compare the action words driven or keyword driven framework against the de facto standard of GUI test automation: a combination of the data driven approach and functional decomposition.
- Actually I never heard/saw a comparison of these two approaches by promoters of keyword-driven frameworks/tools.
- A couple of times during these presentations I asked to compare these two approaches.
- Both times my questions were disregarded!
1st generation of a keyword driven framework

**Conclusion**

**Pros:**
- High Maintainability and High Efficiency
- It’s cool! You feel good implementing such complex framework 😊.
- Respect. Managers do not understand what you are doing. So they respect your skills 😊.
- Job security. Nobody can maintain this framework after you. Managers are at your mercy 😊.

**Cons:**
- Complexity
- Extra overhead
- Not so user friendly (IDE is missing!)
2nd generation of a keyword driven framework

- Worksoft's Certify
- Mercury Business Process Testing

- The same paradigm as in 1st generation applies (script is presented as a table)
- **Integrated Development Environment (IDE) is introduced**
<table>
<thead>
<tr>
<th>Object</th>
<th>Action</th>
<th>Description</th>
<th>On Pass</th>
<th>On Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountNameTxt EditBox</td>
<td>Input</td>
<td>Input @Account_Name into AccountNameTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>ContactTxt EditBox</td>
<td>Input</td>
<td>Input @Contact into ContactTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>AddressTxt EditBox</td>
<td>Input</td>
<td>Input @Street_Address into AddressTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>CityTxt EditBox</td>
<td>Input</td>
<td>Input @City into CityTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>StateList ComboBox</td>
<td>Select</td>
<td>Select the item that is Equal To @State in the StateList ...</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>ZipTxt EditBox</td>
<td>Input</td>
<td>Input @Zip into ZipTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>TelephoneTxt EditBox</td>
<td>Input</td>
<td>Input @Telephone into TelephoneTxt EditBox</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>SaveBtn PushButton</td>
<td>Press</td>
<td>Press SaveBtn PushButton</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>AccountList ListBox</td>
<td>Select</td>
<td>Select the item that is Equal To @Account_Name in the...</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>AccountNameTxt EditBox</td>
<td>Verify</td>
<td>Verify that AccountNameTxt EditBox Is Equal To @Account_Name</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>ContactTxt EditBox</td>
<td>Verify</td>
<td>Verify that ContactTxt EditBox Is Equal To @Contact</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>AddressTxt EditBox</td>
<td>Verify</td>
<td>Verify that AddressTxt EditBox Is Equal To @Street_Address</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>CityTxt EditBox</td>
<td>Verify</td>
<td>Verify that CityTxt EditBox Is Equal To @City</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>StateList ComboBox</td>
<td>Verify</td>
<td>Verify that StateList ComboBox Is Equal To @State - StateList ...</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>ZipTxt EditBox</td>
<td>Verify</td>
<td>Verify that ZipTxt EditBox Is Equal To @Zip</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>TelephoneTxt EditBox</td>
<td>Verify</td>
<td>Verify that TelephoneTxt EditBox Is Equal To @Telephone</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>TotalInvestedTxt EditBox</td>
<td>Verify</td>
<td>Verify that TotalInvestedTxt EditBox Is Equal To @TotalInvested</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>CurrentValueTxt EditBox</td>
<td>Verify</td>
<td>Verify that CurrentValueTxt EditBox Is Equal To @CurrentValue</td>
<td>Continue</td>
<td>Continue</td>
</tr>
<tr>
<td>SaveBtn PushButton</td>
<td>Press</td>
<td>Press SaveBtn PushButton</td>
<td>Continue</td>
<td>Continue</td>
</tr>
</tbody>
</table>

Press the right mouse button for a context sensitive menu.
2nd generation of a keyword driven framework

Conclusion

Pros:
- More user friendly than a 1st generation (Integrated Development Environment vs. EXCEL)

Cons:
- It’s still more complex
- Wrong premise of a Business Analyst used for test automation

When to use keyword driven frameworks:
- When you have a surplus of good Business Analysts (application experts) you do not need for functional testing 😊.
Other alternatives – FIT/FITnesse

- FIT/FITnesse framework:
  - Developed in Java, C#, Perl, …
  - No recording capabilities
  - Wiki (a collaborative tool) is used:
    - To develop test scripts (manually)
    - To report test results
    - To post other comments
- The FIT/FITnesse framework is typically used by agile development teams
FITnesse – sample script

Sample FITness test script:

<table>
<thead>
<tr>
<th>Action</th>
<th>Fixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>fixtures.CountFixture</td>
</tr>
<tr>
<td>check</td>
<td>counter</td>
</tr>
<tr>
<td>press</td>
<td>count</td>
</tr>
<tr>
<td>check</td>
<td>counter</td>
</tr>
<tr>
<td>press</td>
<td>count</td>
</tr>
<tr>
<td>check</td>
<td>counter</td>
</tr>
<tr>
<td>enter</td>
<td>counter</td>
</tr>
<tr>
<td>press</td>
<td>count</td>
</tr>
<tr>
<td>check</td>
<td>counter</td>
</tr>
</tbody>
</table>

- It looks like a keyword driven framework!
Keyword driven framework vs. FIT/FITnesse:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Keyword driven framework</th>
<th>FIT/FITness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data/scripts files format</td>
<td>EXCEL</td>
<td>HTML/Wiki</td>
</tr>
<tr>
<td>Name of executable script</td>
<td>Driver script</td>
<td>Fixture</td>
</tr>
<tr>
<td>Developed with</td>
<td>Commercial GUI testing tools</td>
<td>Java etc.</td>
</tr>
<tr>
<td>Intended for</td>
<td>Independent test team</td>
<td>Development team</td>
</tr>
<tr>
<td>Used by</td>
<td>test automation specialists</td>
<td>developers</td>
</tr>
<tr>
<td></td>
<td>functional testers</td>
<td>customers</td>
</tr>
<tr>
<td>Type of tests</td>
<td>GUI tests</td>
<td>GUI-less / GUI tests</td>
</tr>
</tbody>
</table>

- FIT/FITness is just a variation of the keyword driven framework!
- Designed to be used in an agile environment (no place for an independent testing team)
- **Developers can use the same language to develop an Application and test “fixtures”!**
FIT/FITnesse - recommendations

- We can use FIT/FITnesse in:
  - An Agile environment, when developers are working directly with customers.

- Pros (when using FITness in an Agile environment):
  - FITness Wiki is used to achieve two goals simultaneously:
    1. To document requirements and
    2. To develop test cases
       (Requirements = test cases in this case!)
  - Developers can use the same language to develop applications and test “fixtures”.
3rd generation of keyword driven framework

conclusion

Near future of keyword driven framework:

- More user friendly
  (fully functional modern IDE)
- Better test management
  (flexible scheduling and execution of tests)
- Better test reporting
  (including coverage of data-driven tests)
- Better configuration management/version control

The next generation of keyword driven framework may be able to address the deficiencies of existing frameworks/tools.
6 – A case study of offshore GUI Test Automation
Management issues:

- Management of a team working with 11 hour time shift. Typically status conference calls were scheduled for 8:00 AM EST (7:00 PM offshore time) or 9:00 PM EST (7:00 AM offshore time). It was impossible to resolve any issue at the middle of a work day.

- It was hard to assess the qualification of team members over the phone etc. It happened that not all members of this team were equally qualified to do the job.

- Arising issues were not escalated early enough. Status reporting was improved when direct and specific metrics were introduced. Messages like “80% finished” or “Everything is OK” often were not accurate.
An interaction between functional testers and a test automation team.

- The turnaround for all questions about functional scripts was one work day:
  - Question – today
  - Answer – tomorrow

- Typically questions and answers were in writing; this required more time.

- A person was specifically assigned to provide this type of support – it was his main job (answering e-mails with questions).
Support of test environment/application (at night)

- The test automation team was losing a complete work day when the Application was not available in a test environment because of:
  - Application being down
  - User IDs/passwords having been changed
  - Wrong version of a code of the Application Under Test having been deployed
  - etc.
Network connectivity:

- Getting an approval for SecurID tokens for an offshore team/shipping.
- Responsiveness of a GUI test tool during script development was very slow because
  - an integrated test approach was used and scripts were stored in a test management tool.
  - Network (through a VPN) between the offshore location and the test management tool was chocking scripts development.
Offshore outsourcing of GUI test automation – recommendations

- Do not hesitate to outsource GUI test automation when you already outsource:
  - Functional testing
  - Test environment / Test environment support
- When you outsource only GUI test automation:
  1. Be prepared to spend much more time managing an offshore test automation project.
  2. Assess how detailed your documentation of functional scripts is, and plan/budget for functional support of the GUI test automation team by one of your functional testers / BA’s.
  3. Ensure that you have adequate support for your test environment (e.g. 7/24 pager support)
  4. Ensure that your network is adequate for the demand of your tool / framework.
Offshore outsourcing of GUI test automation – conclusion

- This can be done.
- Risk of failure is significantly higher.
- Be careful and be prepared.
7 – Recommended reading & References
You can find a lot of useful information on GUI test automation on the vendors’ sites:

- Mercury Interactive:
  http://www.mercury.com

- Segue:
  http://www.segue.com/

- Compuware:
  http://www.compuware.com/

- IBM / Rational:
Further reading - 2

- Keyword driven frameworks:
  - “Certify” by Linda Hayes: http://www.worksoft.com/
The following sources were used for this presentation:

- A screenshot of Certify from: http://www.worksoft.com/
The following sources were used for this presentation:

- Sample Perl code from "Automated GUI Testing" by George Nistorica:

- A sample WinRunner keyword driven script by Larry Liu (CGI CTQ)

- A sample FITness script from:
  http://www.fitnesse.org/FitNesse.ActionFixture
Yury Makedonov, CPS
Senior Technical Consultant
Centre for Testing and Quality
CGI services to BCE
125 Commerce Valley Drive West, Floor 6B, Markham, ON, L3T7W4
(905) 762-2705 – office,
(647) 224-6619 – cell
yury.makedonov@cgi.com
http://www.cgi.com