Publication Date: 19 July 2010

Automation Framework and Impact Traceability

By Randy Raymond

Recently, automation engineers at xensight were asked to design an automation framework for use in automated testing of several enterprise applications while simultaneously creating a configuration management plan that avoids regression problems in the automation code itself. The goal was to reduce the maintenance costs of the "write once, use many places" automation code base. The more applications that used the automation framework the more important it became to lock down the automation framework code so that changes made for one enterprise application did not impact the automation suites for the other enterprise applications.

Solution Architects at xensight came up with an interesting solution that also turned out to be an extremely valuable tool for automation code configuration management which dramatically improved maintainability. The inspiration for the solution was taken from the physical product world, specifically automobile manufacture and testing. We have all seen the TV commercials of automobiles being crash tested or seen the documentaries where crash test dummies are used in automobile crash testing. In vehicle crash testing both the vehicles and the dummies are instrumented to measure the forces during a crash.

The solution was to instrument the framework code to measure where it was being used, how often it was being used, and then produce dashboards to report the results. Instrumenting the automation framework was remarkably easy while delivering powerful decision making information.

Instrumentation was in the form of adding a function to framework and other automation components that counted the times the component was used during execution while documenting the application, test case, and business requirement being tested.

To see how the instrumentation would work a picture of the automation architecture needs to be shown. Figure 1 shows the automation architecture.

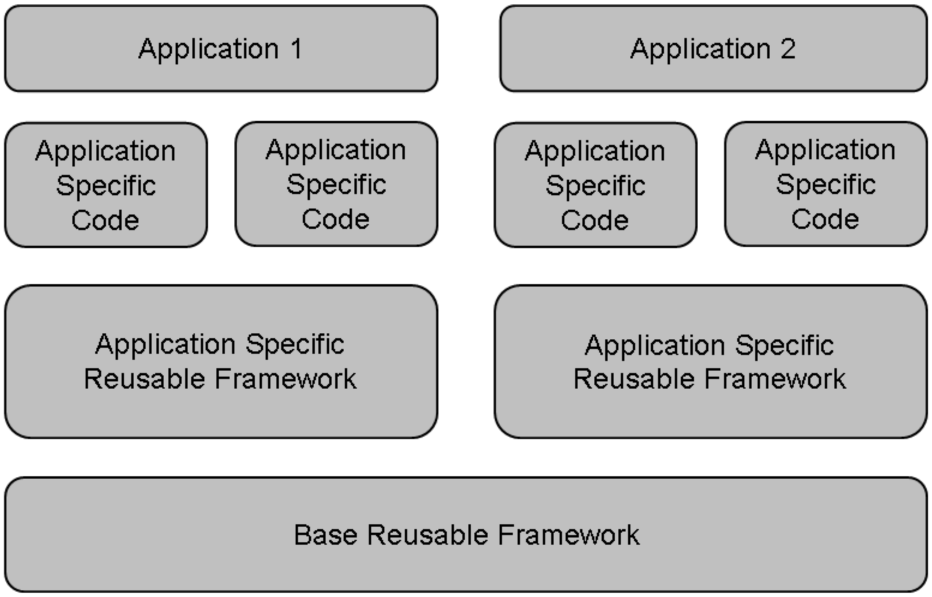

Application 1
Application 2
Application Specific Code
Application Specific Code
Application Specific Code
Application Specific Code
Application Specific Reusable Framework
Application Specific Reusable Framework
Base Reusable Framework

Figure 1. Automation architecture.

Base Reusable Framework (BRF)

The base reusable framework is automation code written to be used across many different applications and is designed to speed development of automation of all applications in the enterprise. This is "write once, use many places" automation code that saves the bulk of time and money when creating new automation. This code is an accelerator but needs to be very carefully managed using tight configuration management policy since one change affects many different places.

Application Specific Reusable Framework (ASRF)

The application specific reusable framework functions identical to the base reusable framework except the ASRF code is directed at one application. Something unique to the application under test requires customized code that cannot be used globally yet can be reused many places when automating the single application. This code makes use of the BRF.

Application Specific Code

The application specific code is automation code written to automate a specific application. It is constructed with both ASRF and BRF along with any custom code needed to automate the application.

Instrumentation

Instrumenting the automation is a straight forward job. The pseudo code shown below provides a very good description how instrumentation was accomplished in our case and can be accomplished in your automation situation.

```
' Global variables

Dim strApp as string              'Application being tested
Dim strTestCase as string         'Test case name or serial number


Function bfw_IsCheckBox_Selected(strObjCheck As String)As Integer
   Call bfw_Log_Script_Usage("bfw_IsCheckBox_Selected", strApp, strTestCase)
On Error GoTo ErrorTrap
    Dim strRetval As Integer
    Dim intRet As Integer
    bfw_IsCheckBox_Selected=0
    intRet=SQAGetProperty("Type=CheckBox;Name=" &
strObjCheck,"checked",strRetval)
     IF strRetval = -1 Then bfw_IsCheckBox_Selected=1
  Exit Function
ErrorTrap:
   Call fw_trapError("bfw_IsCheckBox_Selected" , Err, Erl)
End Function
```

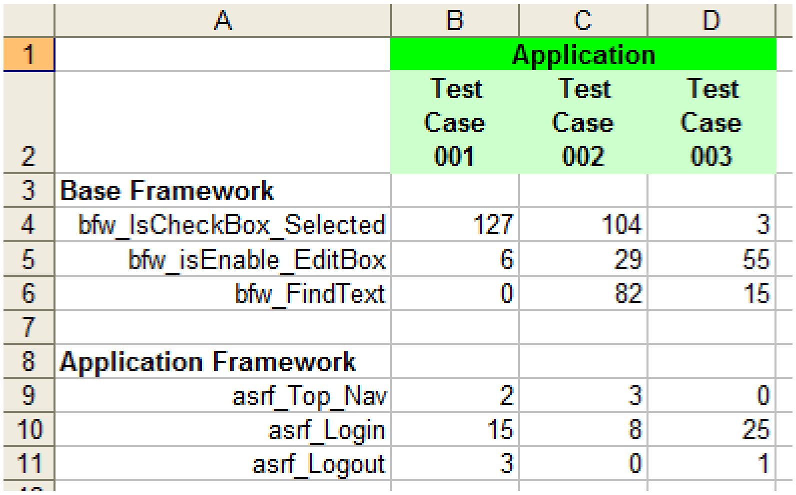
The function bfw_IsCheckBox_Selected is a BRF function that inspects a checkbox to determine if the checkbox is selected. The automation tool is Rational Robot and this strategy will work for any of the market leading test automation tools.

Each function and subroutine in the ASRF and the BRF calls the subroutine bfw_Log_Script_Usage which records the name of the function or subroutine being executed, the name of the application under test, and the test case that was being executed. The resulting information logged during test suite execution provides the following;

- Number of times a BRF component is used during execution of a specific test case
- Number of times a BRF component is used during execution of all tests in an application
- Number of times an ASRF component is used during execution of all tests in an application

- Number of times an ASRF component is used during execution of a specific test case

The log in a very simplified spreadsheet may appear as shown in figure 2.



	A	B	C	D
1		Application		
2		Test Case 001	Test Case 002	Test Case 003
3	Base Framework			
4	bfw_IsCheckBox_Selected	127	104	3
5	bfw_isEnable_EditBox	6	29	55
6	bfw_FindText	0	82	15
7				
8	Application Framework			
9	asrf_Top_Nav	2	3	0
10	asrf_Login	15	8	25
11	asrf_Logout	3	0	1

Figure 2. Simplified log.

Traceability and Impact Analysis

Figure 2 is very simplified and yet it shows a very valuable picture of how to reduce automation suite maintenance regression problems through traceability and impact analysis.

Individual reusable framework modules have been traced to a specific test case and a specific application. See figure 3.

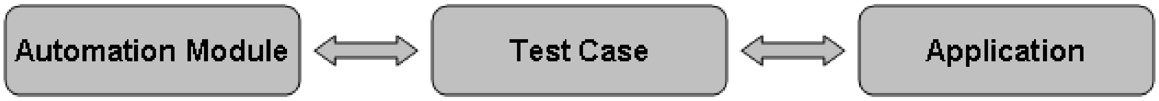


Figure 3. Framework module traced to application.

If you are following best practices on requirements traceability to test cases you can trace each framework automation module to an application requirement. See figure 4.

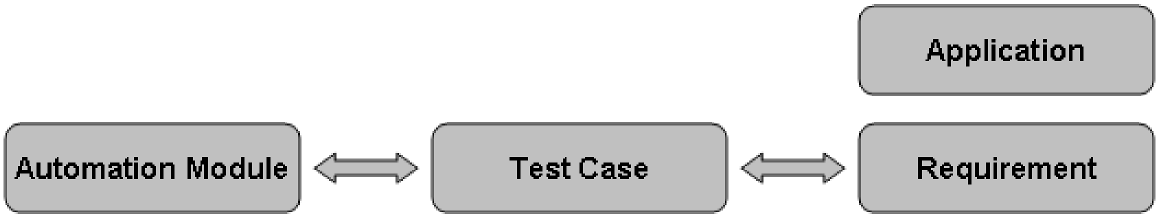


Figure 4. Framework module traced to requirement.

This traceability arrangement means that if a requirement changes then corresponding test case changes can be identified and ultimately automation modules used in the automated testing can be identified.

Single Application Impact Analysis

The basic measures described in the Instrumentation section above can be used for impact analysis with respect to configuration management of the automation code. Disciplined configuration management of the automation code leads to lowered maintenance costs.

For example, if a test case is changed, the change might necessitate a change in automation code. If that change is in the application specific automation code then the change is easy, you simply make the change. If that automation code change is in the BRF or ASRF code a change in either of these two areas can impact a large number of test cases that have been automated. The automation engineers managing the code configuration can perform traceability research to see what other test cases will be impacted with the needed automation code change. The results of the traceability analysis will determine if the BRF or ASRF code can be changed or if another code solution is warranted.

The big benefit is in maintainability and automation suite reliability. The configuration management team can make informed decisions on where to make changes. The regression problem of making a change in one place causing something to break in other places is materially reduced since the configuration management team is making informed decisions by knowing all test cases that are impacted before changing any automation code. In our example, we are avoiding making an automation code change to accommodate a modified test case and having that automation code change break our automation in one or more places elsewhere.

Multiple Application Impact Analysis

Configuration management of BRF is much more important since this automation code is used to construct automated tests for more than one application. Changes to the BRF automation code should not be made without thorough understanding of the impact so as to not introduce maintenance problems in all of the test automation for the several applications using the BRF. See figure 5.

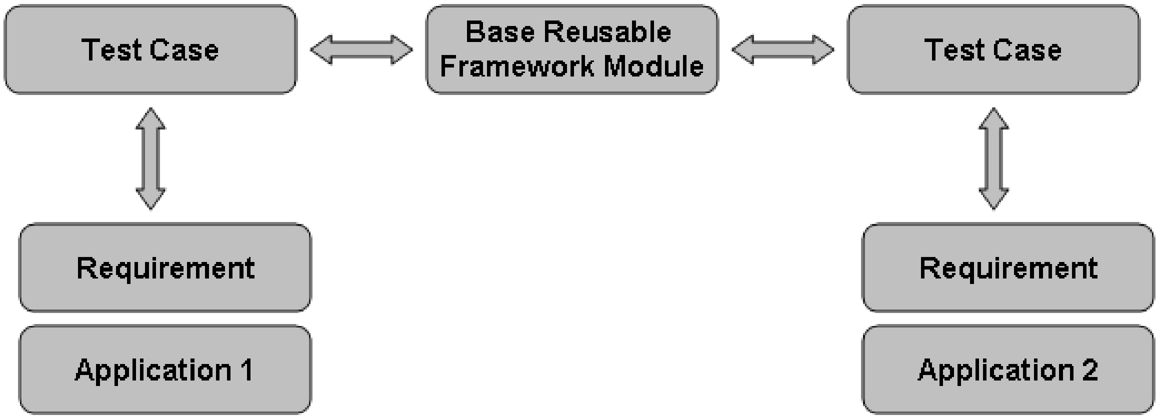


Figure 5. BRF multi-application impact.

Downside to Instrumentation

The downside to the approach described in this paper is that the numbers start getting very large as you execute your test suites over time. This will be particularly evident in the base framework components that have been used to construct many other pieces of your automation. Over a couple of years some BRF components will have been used during execution millions of times.

A second, and probably the most important downside, is that the data source where you choose to send your log data needs to always be available when you run your scripts. Most of the time the data store will be a database of some kind. Your automation suite will need connectivity to all enterprise applications under test *and* the data store.

Lastly, since your instrumentation is constantly writing to a data store the execution of the test scripts themselves will be slowed down. Execution time will be expended incrementing a number instead of directing a test at the AUT.

Like all technical activities there will be tradeoffs. You will need to decide for yourself if the management information and control over the automation framework is worth accepting the downside complexities.

What We Have Omitted

We have omitted on the details of the logging function, `bfw_Log_Script_Usage`, in our example above. The design of the logging function is entirely based on your specific situation which could include;

- The automation testing tool is being used
- The logging data store - text file, spreadsheet, Access database on the local machine, other database located on the network, etc.
- The form and location of your requirements trace matrix (RTM) that traces application requirements to test cases
- How to relate the logging data store to the RTM
- How traceability is reported for one application like our simplified figure 2 and how traceability is reported across multiple applications

The design of this function, the data store, how it relates to the RTM, and how traceability is reported becomes a critical success factor. You should spend a significant amount of time planning this architecture then be perfect in your implementation.

Conclusion

Retaining the promised value of creating a test automation framework involves significant configuration management and change control discipline over the framework itself. The more applications that are automated using the framework the tighter this configuration management and change control discipline must become in order to reduce regression defects - i.e. reduce maintenance costs - in the automation code.