Ghost in the Machine

John Kent, MD of Simply Testing Ltd, continues his series.

Part 4: Advanced test automation architectures

A test automation architecture is the same as a software architecture but built in the language of the test tool. But what is a software architecture?

What is an architecture?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" –Software Architecture in Practice, Bass et al, (Addison-Wesley 1997)

So, a software architecture is the structure of the source code and the relationship between the components. In civil engineering, the architecture is the overall structure of the building. The architecture provides the space for the functional use of the building and ultimately is the major factor affecting how successful it is at fulfilling its purpose.

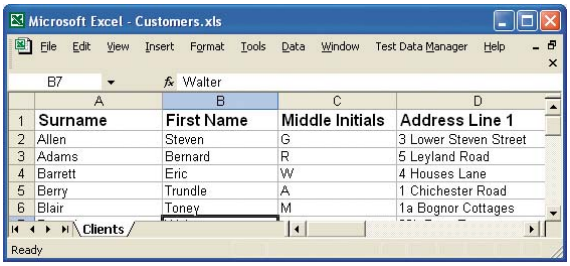

Figure 1: Simple 'data-driven' data

We have already seen that scripted test automation consists of code and is really a product of software engineering, therefore it should have a good architecture to be successful. Little wonder that test automation does not get very far when left to junior programmers or testers when they have some spare time. No matter how hard-working these people may be it is too much to expect them to create a good architecture. They are often too busy with the low level detail to be able to see the overall structure and do not have the software engineering experience to design architectures. They are builders, not architects.

A good test automation architecture provides structures for logging and error reporting and allows recovery to be incorporated (dropping failed tests, navigating the system to a base point and continuing with the next test). It will have libraries of reusable functions in much the same way as any other software system. Most importantly of all its structure will minimise the code maintenance overhead and thus it is the starting point for success with test automation.

Advanced test automation architectures are really the logical place to go when you take a software engineering approach to building test automation. So how do advanced automation architectures differ from the basic data driven approach?

In order to illustrate this, let's first look at some data used in data driven automation (see figure 1). In this example of Excel test data, every line (except the heading line) has customer details in it. Column 1 is the surname, column 2 the first name etc.

In the test tool there will be a function which repetitively reads in this data and inputs it into the 'Add customer' windows or screens. Most basic data driven architectures for more complex systems involve many different data files for the different types of business functions and many automation programs to input the data.

One of the main disadvantages of the basic data driven approach is that the automation pumps data into the system in an unnatural way. Most regression tests are not about repetitively inputting similar data into the user interface; rather they seek to fully exercise all of the business functionality of the system under test in a realistic way.

Automated testers have recognised this for a while now and some provide a solution for it by adding in another layer to the automation which specifies the order in which data is used from the various files. For example in an insurance system, the extra layer would select the customer data to add, then select the insurance policy data to add for that customer and so on. This however becomes a mess architecturally.

What the advanced architectures do is take (at least some) of the navigation and actions out of the test programs and put them into the test data. The test data becomes the script—the sequence of actions to be followed. It tells the automation code what to do and in what order. Advanced architectures allow the test analyst some choice about what the sequence of events should be. The level of choice is dependent upon how sophisticated the architecture is.

Figure 2 illustrates data from an advanced approach. Note that if there is a hash at the beginning of the left hand column this means that the line is a comment and not test data. Lines beginning `Supplier_Add` or `Stock_Item_Add` are the actual test data which will be read by the automation code. The format of the file is very different to that in figure 1 because the meaning of the data in each column is dependent upon what type of line it is and this is defined in the first column. The comment lines give the format so that the test analyst knows what each column represents. For example in the seventh line, column four is the 'Name' field because this is a `Stock_Item_Add` format. Thus the test analyst can choose the order of the actions when creating the Excel data.

#Supplier_Add	Sup Name	Description	Type	External Flag
Supplier_Add	SC_SUP00	SC-SUP00 Supplier Company		OFF
Supplier_Add	SC_SUP01	SC-SUP00 Supplier Company		ON
#Loc_Add	Loc Name	Description	Corp	Corporate Y/N
Loc_Add	SC-SXX-1	Test LO Stock Room only	ON	OFF
#	SI Name	Item Description	Name	External Flag
Stock_Item_Add	SC_STCK00	CABLE	COAX	Test SI
Stock_Item_Add	99	PUBLICATION	BOOK	Wind in the …
#	SI Name	Description	type1-01	Quantity
Stock_Loc_Add	SC_STCK00	SC-SXX-1	1-01	10
Stock_Loc_Add	99	SC-SXX-1		10

Figure 2: Advanced architecture data

When the automation runs, a driver program reads through the data and calls the function specified in column 1 passing it the data for that particular line. For example there will be a function `Stock_Item_Add` which will have been written by the test programmer (scripter) in order to perform all of the actions

required to add a stock item. These functions are what are known as *wrappers*.

Automation wrappers

Wrapper is an OO term that means some software that is used to interface with an object. In programming terms you call the wrapper if you wish to use the object. You can't call the object directly.

In test automation, wrappers are programs or functions written in the language of the test tool which perform discreet automation tasks as instructed by the test data and actions. They provide the interface between the test and the system under test's user interface. In our previous example there are four wrappers – `Stock_Item_Add`, `Loc_Add`, `Supplier_Add` and `Stock_Loc_Add`. These are business level wrappers – they 'wrap' business functions and were written by the test programmer (scripter). You will be able to see that the example in figure 2 is similar to the data driven approach but the first column in the test data of each line tells the automation which wrapper to call.

There are two distinct types of advanced architecture. Figure 2 shows a *business object level architecture*. In this type of automation architecture, one automation function or program is written in the test tool for each type of business task. These functions are the wrappers for the business tasks.

Usually a functional decomposition of the system is the first step in building this type of architecture. In a functional decomposition, the basic business functions of the system are defined. Then the wrappers for each business task are programmed (scripted) and the data format for each task is created.

#Screen					E&xit		Fo&rward					
#	EditBox	POLREFIN	EditBox	GRPREF	Button	cmdBUTC	Button	cmdBUTTWO_DATA				
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA				
ZB01N				Scheme			CLICK					
#Screen									Employed contribution		Contracting out of SE	
#	EditBox	SCHNAM	EditBox	CLIREFIN	E	POLREFIN	EditBox	SCHINQ_D	CheckBox	chkEEE_D	CheckBox	chkSRP_D
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
ZB15N					CHECK	AD012083			SETON		SETON	
#Screen												
#	EditBox	SCHINQ_D	EditBox	CLINAM	e	PRDNAM	EditBox	SCHNAM	e	CLISUR_D	EditBox	CLIFOR_D
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
CL12N										Kent		John
#Screen												
#	EditBox	SCHINQ_D	EditBox	CLINAM	ListBox	PRDNAM	ListBox	SCHNAM	ACT	PCDTWO	ACT	PCDONE
#	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA	ACTION	DATA
CL02N						PWF03				4TP		GU21

Figure 3: Screen\window level advanced architecture data

Test data in business object level architectures is at the business language level and therefore understandable by end users, which is a great advantage.

Screen/window level architectures

In this architecture there is one test program that deals with each screen/window in the system—it acts as a wrapper for that screen/window. It handles all of the input and output (getting expected results) for its window or screen. See figure 3 for an example of the data. Again lines with a hash in the left hand side are comments used to show the test analyst the format of the data for that screen/window. The other lines are the test data which is passed to the automation wrappers for that particular screen. The first column of the test data represents the screen that the data refers to and thus the format of the line is dependent upon what the screen name is in column one. Also note that, as this a test of a GUI system, for each user interface object there is an action and a data field. Thus the test analyst can specify actions like 'Check' that the object contains the data equal to 'Smith' or even '`CheckEnabled`'.

One of the biggest advantages of screen/window level architectures is that all of the user interface objects can be made available to the test analyst in Excel and thus the analyst has complete control over what the navigation and actions should be, rather than being dependent upon what the test automation programmer (scripter) has put into the wrapper—as with business object level architectures.

Next issue: New ways to create automation code

PT