Ghost in the Machine

Part 6: New Ways to Build Test Automation Code

This is the last article in the John Kent series on software test automation and discusses where test automation is heading. The previous article showed that for medium to large systems, even with the most maintainable automation architecture possible, you still have thousands of lines of automation code to build and maintain. This has prompted some automation specialists to look for easier ways to build automation.

John Kent
Managing Director of Simply Testing Ltd and a specialist in test automation.

Map Based Automation

One approach is to remove programming (scripting) of automation code from the process altogether. You may have heard of what are often called automation engines that do this. The way to achieve this is to have a driver program that interprets a 'Map' of the user interface. We haven't discussed maps much in this series. Many of the tools separate out the user interface object definitions from the actual scripts or code and put them in a 'Map'. The object identifiers - the way the test tool recognises which UI object is which - are defined in the map. The scripts or code use a symbolic name for each object.
WinRunner, for example, has the GUI Map, which contains UI object definitions. WinRunner TSL code uses the names of these mapped objects rather than the actual object identifiers themselves. This helps with maintenance, for if a window or an object changes, only the map will need to be changed. Up to a point anyway.

It is possible to build an automation 'engine' which, when fed with instructions from test data, recognises which object in the map is to be acted upon and then performs the specified action. If your test tool does not have explicit Map functionality you can build it based upon object identifiers and names in a separate data file. Look at the test data shown below. The automation engine reads in each line of this data. It knows which window the instruction is for from the wrapper name and it has the name of the object within that window. The actual way the object is identified is defined in the Map, so the automation engine does not need that information, only the name of the object. The automation engine then performs the action on the object, using the data if required. No automation code specific to an individual object is needed. The only code required is the automation engine functions, which interpret the test data and perform the action.

An important thing to note here is that the number of lines of code in your automation engine is fixed. It is not proportional to the number of UI objects in a system or even the number of windows. It is a constant. Not only can you create more tests without having to build automation code, you can define more windows/screens/web pages in the GUI map and not have to increase the number of code lines at all. All you have to do is define the new windows in the Map. In fact, you could use the same automation engine on many different systems without having to write more code. Thus maintenance of this automation only involves keeping the map and the test data up-to-date with the changes to the system under test. Nice. Better than nice.

Automation Generators

We mentioned in the last article a largish system, which had around 900 windows.

Wrapper	Object	Action	Data
Login	UserID	SET	user1
Login	Password	SET	password
Login	OK	CLICK	
MDIMain	Open	CLICK	

Using the automation engine you would need to map 900 windows to get 100% automation coverage. This can be quite a large task and maintaining the maps can take a lot of effort, so tools have been developed that can generate the maps automatically. The UI object definitions are read by the generator automatically from the system under test and the maps can be generated extremely quickly. They can be regenerated for each release of the system thus vastly reducing the maintenance task.

Data Starting Positions for Automation

A word about data for systems based on databases. This topic could take up a whole article itself but we don't have the space here to do it justice. We do need to mention it, however, as it has a bearing on how you build your automation. With window wrappers (see article 4) and map-based automation approaches you generally assume that you can predict what the data are going to be when the automation run starts. You cannot just run automation against any old database.

The Excel spreadsheets in advanced automation approaches are scripts. The Excel data consist of a series of detailed instructions on what to do, and usually requires data to be already set up in the database. For example it is no good trying to create a bank account for a customer when the customer does not exist in the system's database. This is a trivial example, but the situation for most database-type systems is very complex and you must decide on what data you have in your database at the start of an automation run. You may decide to start with an empty database and have the automation create all the data it needs. This, however, may mean that you get bogged down in creating masses of complex reference data even before you start to create your regression tests. The alternative is to use a database that has known data on it and always restore the database back to this known point before you start your automated test run. In the example just given, this restore would contain the customer your test needs. The important thing is the starting position. Your automation data assumes a known starting position, an empty database, or known data. You may think this is a trivial point but it is very often the thing that causes the failure of an automation project.

Sometimes you can't do database restores. Perhaps the database interfaces with too many other systems to make this economical. In this case you may need to build functions to go and search the data for some suitable data to use in your tests. These would be similar to the business object wrappers mentioned in article 4. You could use a mixture of business wrappers and window wrappers or map-based automation.

Series Summary

If I had to find a word to sum up the message of this series of articles on automation it would have to be: Maintenance. If you can't maintain your automation code fast enough to test the next release of your system, you might just as well not bother. The scripted test tools we have today are not a complete solution in themselves. You have to build code in order to automate tests, whether using record/playback or programming.

You then have to keep that automation up-to-date with the different builds of the system. With the wrong automation approach, maintenance is just too difficult ■

johnkent@cisstest.com