Ghost in the Machine

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

Part 5: Why we need new ways to create automation code

How much scripting do we have to do in order to build an automated regression test pack for a system? Say we wish to create tests that cover 30% of system functionality. This may seem quite low, but accept it for now.

Let's look at some relationships in order to illustrate the size of the programming tasks for the various different approaches. These are not equations; rather we are trying to see what parameters play a part in determining the amount of programming necessary for an automation project. One of the measures that we are going to use here is the number of screens or windows in the user interface. This gives us a measure of the size of our system.

Here's our first relationship. In record/playback or non-data-driven program/playback, *the amount of scripting needed is proportional to the number of tests and also the number of windows in the SUT.* This can also be written

Amount of code ∝ number of tests and number of windows

"Windows" here might mean screens, windows or web pages.

What this is expressing is that for the non-data-driven approach the number of lines of automation code you need to build is dependent on the number of tests. This is because each test is 'translated' into the language of the test tool. Therefore, the more tests you wish to automate, the more automation code you need. I think that there is also a relationship between the number of lines of code and size of the SUT's user interface: if there were only one window in the SUT the automated tests would be quite short and, conversely, if the user interface has 1,000 windows the tests would generally be longer because they would traverse more windows.

As soon as you move to a data-driven approach you break the relationship between the number of lines of code and the number of tests. With data-driven automation it is possible to increase the number of tests by simply adding test data and not necessarily having to increase the amount of automation code.

Now consider Business Object Level Architectures. *(See previous articles)*

Amount of code ∝ number of business processes and number of windows

In Business Object Level Architectures automation programs (or functions) are built for each business process that is to be automated so the numbers of processes must be a factor in the relationship. Again, those processes are probably proportional in size to the overall size of the system's user interface.

Finally consider screen/window level architectures.

Amount of code ∝ number of windows

> "the number of lines of automation code you need to build is dependent on the number of tests because each test is 'translated' into the language of the test tool"

In screen level architectures there is one *wrapper* (or function) for each window and no other automation code interfaces with the user interface objects. Thus we have a very simple relationship here, and we are even quite close to estimating the actual number of lines of automation code for screen/window level architectures. Because the amount of code is directly proportional to the number of windows in the SUT the equation (as opposed to relationship) to give the amount of code must be:

Amount of code = k(number of windows)

Where k is a constant. Note that this is now a real equation rather than a proportional relationship.

k is made up of two things. First let's consider another measure. This measure of your system is related to the number of windows in the SUT: it is the total number of *user interface objects* (buttons, listboxes, comboboxes etc); also called controls, widgets etc depending on the platform. A text-based system has only one type of UIO; a *field*. From this number we can calculate the average number of UIOs per window in the system.

We are almost there now. The other component of k is the number of lines of automation code per user interface object (LOC per UIO). This is the average number of times the automation code refers to each UIO.

So now we can put this into the equation to give the following:

Lines of automation code = w × u × l

where w is the number of windows in the SUT, u is the average number of UIOs per window, and l is the number of lines of automation code per UIO.

This may seem like a circular argument: *the amount of automation code is dependent upon the number of lines of automation code per user interface object*. However, for well-designed automation screen/window level architectures this number can be determined and is always the same; for the WinRunner version of of Simply Testing's automation framework *ATAA* it is around 8.

In fact the equation can be simplified. The total number of UIOs is given by w × u, so:

Lines of automation code = total UIOs × l

Or: *the number of lines of automation code equals the total number of UIOs multiplied by the number of lines of automation code per UIO.*

We have not included any automation code that is not specific to the user interface. Automation architectures or frameworks have to include code to write out results and logs,

input the test data, etc; let's call this *f*. So the final equation is:

Lines of automation code = total UIOs × *l* + *f*

Let's put some numbers into these equations and see what we get. Let's say you have a system with 900 screens in it; large, but not unusually so. You may be able to measure the total number of user interface objects in your system, but for those that can't, let's take the average number per window. Say 20 objects per window/screen (this is low). We get 20 × 900 = 18,000 user interface objects in the system. Assuming 8 lines of automation code per user interface object, we have a grand total of 8 × 18,000 = 144,000 lines of automation code required for the entire system.

At the beginning we said we would aim for 30% coverage so the number of lines is reduced: 144,000 × 3 = 43,200 lines of automation code in WinRunner, QARun or whatever tool you are using. Also remember we have not included the code for I/O, the drivers and library functions (f). In my experience this is usually around another 7,000 lines.

So that's 50,200 lines of code to build and, more importantly, to maintain. Extremely good programmers can build 50 lines of code per day so now we have an estimate of how many scripter days we need to build the automation. It is about 50,000 ÷ 50 = 1,000 days. This is for only 30% system functionality. If you were aiming at 100% coverage it would take well over 3,000 days.

The limits of program/playback

Just as we hit limits with record/playback where it becomes impossible to maintain the huge amounts of automation code generated for a regression test pack, we hit limits with program/playback. These limits are limits in terms of being able to build the automation pack, and limits in terms of being able to maintain it.

We can see from the equation that, for small systems we don't hit these limits. I've seen trading systems which have very small numbers of windows (around 10-15 windows). Call it 15 windows and again assume 20 UIOs per window and you get 20 × 15 × 7 = 2,100 lines of automation code. This is manageable and maintainable within many test budgets.

Most of the systems we are testing however, are very large. I hope this equation illustrates why program/playback hits limits for large systems. It also shows why automation often disappoints, because program/ playback cannot give the test coverage people hope for. Think about the size of your system before you start automation and the amount of functional coverage you are aiming for. Then estimate the amount of automation code you need to achieve this.

We can't determine the equation for business object level architectures because we cannot know precisely how many windows or screens are traversed during a typical business process. However the same principles apply but perhaps more so because Business Object Level architectures are less efficient in terms of LOC per UI object than screen level architectures.

Looking at these numbers, it is quite obvious that for large systems we need to find easier ways to build test automation code. *We need to automate the building of the automation code.*

Next issue: the new ways to build automation code reviewed

PT