The view from

John Kent begins a new series on test automation

Ghost in the machine: automated testing Part I: The myth of record/playback

Introduction

Software test automation seems to be on every test manager's list of things to have. Every test organisation wants it. The automation tracks at conferences are usually the best attended and most of the large test tool vendors have made themselves large by selling tools to do it. It is often, mistakenly, the first thing people start with when they embark upon improving their testing procedures. All of this illustrates how attractive test automation is - not how easy it is.

The attraction to test automation is strange when you consider how little success it has had. How many automation projects return real benefits for their investments? In automation circles there is a debate about why the success rate is so low. Is it because the wrong tools are being used? Is it because of a lack of management commitment? Is it a failure to manage expectations? All of these things are important, but much more important than these is the fact that the technology and the way it is used is immature.

Scripted tools running under Windows have only been around for a short while - Microsoft Test (now Visual Test) version 1.0 came off beta test in early 1992, about the same time as the other major tools we now have. The technology is difficult to use, but new approaches to their use are evolving and it is an exciting time to be involved in this area. One element in these new ways to build test automation is a software engineering approach to the automation architectures.

We will discuss architectures in a later article but first let's look at the biggest myth in software test automation: record/playback.

Just before we do that it is worth underlining what we are talking about here. The script-based tools such as Robot, SilkTest, WinRunner and QARun are used to play actions-key stokes and mouse clicks-into user interfaces. We are not talking about what are commonly called test harnesses used by programmers at unit test level. The tools we are discussing are generally used by test teams to automate system and regression tests.

Record/playback

Automated testing tools are often called 'capture/playback' or 'record/playback'. It sounds great doesn't it? "…all you have to do is simply record your tests and replay them whenever you like." It is a sad fact that this is not the whole truth. Record and playback looks great, but it only really does what is it is supposed to do in one situation: the tool vendor's sales demo. It is the ideal sales feature - you will see a great demonstration of a software robot doing the work a highly paid technician used to do. It's simple and you don't need expertise to do it. Wonderful, I'll have four please.

Record/playback means recording actions against your system's user interface in a 'script', and then replaying it back into the system under test (this does not apply to mainframe tools, which are not script based). The recorded script will be a series of actions in the tool's language. When you replay the script, the tool replays the actions back into your system under test's user interface. As an example, let's try a live demo - well at least as 'live' the written page will allow:

I'll record myself typing in the next sentence using Visual Test. Here we go: Recording also produces unmaintainable scripts that consist of a long list of references to objects on your user interface.

I've just stopped the recorder and this is what the recording (the script) looks like:

```
CurrentWindow = WFndWndC("Microsoft
Word - TVFK.rtf", "OpusApp",
FIND_AND_MAX, timeout)
Play "{Click 538, 0, Left}"
Sleep(2.374)
Play "{ENTER}Recording also pr"
Play "oduces
BS}unmaintaai{BS}{BS}inable scripts"
Play " that"
Play " "
Play "consist of a long list of refer-
ences to objects on your ue{BS}ser
interface"
Play "."
```

Kent

The '{BS}'s are where I have typed a backspace. I'll try to play this back (it should type the next paragraph) and see what I get:

Recording also produces unmaintainable scripts that consist of a long list of references to objects on your user interface.

Brilliant, so it works. So now I'll record underlining the word *unmaintainable* here and italicising it by selecting it with the mouse and clicking the Underline and Italics buttons. This is what I get when recording with Visual Test:

```
Play "{BtnDown 409, 136, Left}"
Sleep(5.232)
Play "{BtnUp 513, 137, Left}"
Sleep(2.106)
Play "{Click 302, 53, Left}"
Play "{Click 320, 53, Left}"
```

I tried to replay this and eventually, after some editing of the script, got it to work. I had to remove the first sleep statement in order ensure that the word was selected and of course I took the underline and italics off in order to return to the initial state prior to running. It worked after editing, but look at this script. Not only is it unpleasant code, it is also unmaintainable - well, at least very difficult to maintain - because it has XY co-ordinates of mouse-clicks and it is therefore almost impossible to identify what it is doing to which user interface objects.

To be fair to the tool vendors, they have improved their recorders so the user interface object on which the actions are performed is identified in the recording for most of the common user interface development tools. Below is a fragment of Rational Robot recording where an 'OK' button is clicked and a File|Exit menu item is clicked.

```
PushButton Click, "Text=OK"
MenuSelect "File->Exit"
```

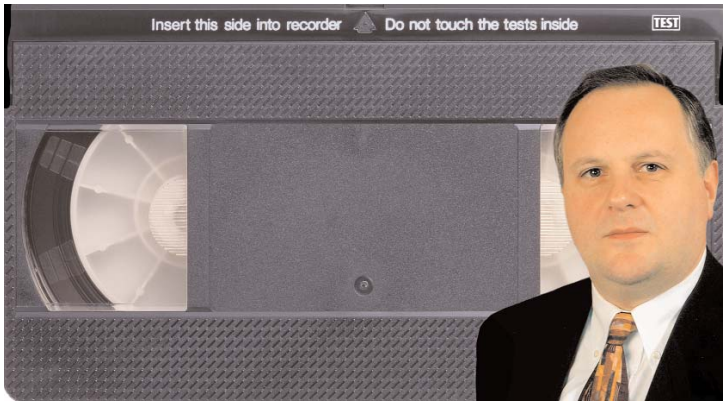
In WinRunner TSL the same actions look like this:

```
button_press ("OK");
menu_select_item ("File;Exit");
```

In QARun they look like this:

```
Button "OK", 'Left SingleClick'
MenuSelect "File~Exit"
```

These are all better than the XY co-ordinate mouse-click situation but they are still not good enough to make record/playback practical for large-scale automation.



The code fragments look maintainable and they are - but this maintenance is only practicable on a small scale.

We are at the most important issue in automation here. It is this issue of *maintenance* that is key to the success of test automation projects. With automation you are interested in testing the system as it is changed. The user interface and behaviour of the system changes over time and you must change your automation to work with the changed System Under Test (SUT). In test automation you are interested in subsequent releases of the SUT because you have already proven that the system does what you've recorded. The payback with automation starts when you test later releases of the system.

Yes, you can maintain the above automation code (script) fragments. You can even maintain the first script with its XY co-ordinates. Yes it is *possible* to maintain scripts like these but is it practicable? How many lines of script do you think you will need for a system test or regression test? That is very much dependent upon how many windows/screens your system has and how many tests you want to perform. My current client has 949 screens in their system. A while back one of my clients had an automated regression test pack consisting of 70,000 lines of recorded automation code. That's 70,000 lines of code to maintain as the system changes - a huge task - in fact too big a task for most test organisations as my client discovered. You could maintain the code but it would take such a huge team that it would be prohibitively costly.

Thus with record/playback, the more tests you record, the more automation code you have to maintain until you reach a point where the maintenance burden becomes too

The myth of record/playback

great for your budget. So it is the really the cost of maintenance that is the key issue governing the success or not of software test automation. Yes, it is *possible* to maintain recorded automation code but it is the cost of doing this for large recorded test packs that makes it impractical.

We haven't finished putting record/playback in its place yet. There are more arguments against using it for large numbers of tests, the most important of which is that re-runnable tests must cope with what *may* happen, not simply what *has* happened.

For example, one of the objectives of running tests against a system is to find errors. The automated tests must have code which recognises errors and this must be programmed into them. If they don't have this intelligence and an error is encountered during a test run, the whole test schedule will come to a stop. It is indeed ironic that, although the objective of automated testing is to find bugs, if the SUT behaves in an unexpected way - i.e. there are bugs - then these automated tests will not work.

Thus a test cannot simply be a recording played back into the System Under Test (SUT), rather it needs to be an interaction between the test tool and the SUT. You can use recording as a way of helping to create automated tests but you will have to take them and modify them by programming. Record/playback tools are not really Record/playback at all; they are program/playback tools.

There are still more arguments against record/playback. There are problems with synchronisation and object identification. Perhaps the best demonstration of what is wrong with it can be seen if you try recording just one hour of input with one of the modern script-based tools against your system. Take a look at the recording and see if you think you could maintain just one hour's worth. Then run it and see how well it replays. Without knowing anything about your system I can guarantee that it will not run all the way through. Record/Playback may be OK for knocking together short, simple demonstrations but for large, real-life automated regression tests it is a non-starter.

John Kent
Managing Director of CISS Ltd
who specialise in test automation