AUTO101: An Introduction to Test Automation Techniques

Test Automation Techniques

Welcome to our Introduction to Test Automation Techniques!

We'll start the tutorial describing an "Ideal" automation framework. Then we will describe some common automation techniques used today, and go over how they might fit in with our "Ideal" automation framework.

But first, tell us a little about yourself. Let us know your expectations in the Quiz/Survey and Journal below.

Your Test Automation Background

Your Expectations and Comments

Terms for SAFS Test Automation

Survey: Your Test Automation Background

To enable us to provide a better online training experience, we would like to survey your opinion of your own test automation expertise. By understanding the experience level of those taking this course we can continually improve the content and presentation to better serve the needs of our students.

When you are finished with the "quiz" and returned to this screen, click on the **AUTO101** link at the top of the page to continue the tutorial.

Thanks for your time!

1 How long have you been working with software test automation? We are interested in your accumulated experience solely on test automation. So, if you've been testing for 1 year, but only 50% of your time has been working on test automation, the appropriate answer would be *6 months*.

Answer:
- ○ a. Less than 3 months
- ○ b. 3 <6>
- ○ c. 6 <12>
- ○ d. More than 12 months

2 Have you ever used *Record and Playback* scripting before? Record and Playback scripting is when the automation tool is turned on to record user actions (mouse and keystrokes) and it captures these into a test script. The script can then be played back for test automation.

Answer:
- ○ a. Yes, it has worked great for me
- ○ b. Yes, but I had problems with it
- ○ c. No, I have never tried it
- ○ d. I don't understand what you mean by Record and Playback

3 Have you ever used *Functional Scripting* for test automation? Functional scripting is when available automation activities are broken down into functions and subroutines that can be called as needed from another test script.

Answer:
- ○ a. Yes
- ○ b. No
- ○ c. I don't understand what you mean by Functional Scripting

4 Have you ever used *Data-Driven Scripts*? Data-Driven scripts are used most often to accomplish a highly repetitive task with varying data values. For example: filling an address book database with thousands of entries.

Answer:
- ○ a. Yes
- ○ b. No
- ○ c. I don't understand what you mean by Data-Driven Scripts

5 Have you ever used *Keyword-Driven* Tests? With Keyword-Driven testing--which is also known as Action-Based testing, or Action Word testing--the automation tool script no longer contains the instructions of what to test. Instead, the tool script becomes a generic parser or test interpretter that reads external commands (keywords) and performs test actions accordingly.

Answer:
- ○ a. Yes
- ○ b. No
- ○ c. I don't understand what you mean by Keyword-Driven Tests

6 Have you had any success with test automation in the past? We know that test automation success is never assured. Some projects work out, and some don't--for many different reasons.

Answer:
- ○ a. Yes, I have been pretty successful with automation.
- ○ b. Mixed results. Sometimes good. Sometimes bad.
- ○ c. No, I have had nothing but problems with automation.
- ○ d. No, I have not yet been involved in any automation project.

7 Which tool vendor provides the tool or tools on which you have the most test automation experience?

Answer:
- ○ a. IBM Rational
- ○ b. Mercury Interactive
- ○ c. Seque
- ○ d. Compuware
- ○ e. Empirix
- ○ f. Open Source Tools
- ○ g. Proprietary In-House Tools
- ○ h. Other
- ○ i. No experience yet

Terms for SAFS Test Automation

A

Application-Independent: The framework or tool is not exclusively designed to work with just one application. It is intended to work across many applications, or all applications, without modification. Simply, it was made to test all applications, not just one.

Automator: See "Test Automator"

D

Designer: See "Test Designer"

Domain Expert:

The "domain" is the context or subject matter of the application being tested. Example: An application for accounting or finances targets the domain of users wishing to do accounting or financial operations.

A domain expert is an individual that has expertise in the field the application is designed to satisfy. They are the individuals that review or test the application with the perspective and knowledge of the application's customers.

F

Functional Scripting: The test technique in which automation tool scripts, or script libraries, are broken down into callable functions or subroutines. These functions provide simple reusable actions or features that can be called many times by external scripts or script libraries, usually with different parameter values to provide variation on the function outcome.

K

Keyword-Driven: A test technique in which tests are expressed with user-defined keywords and actions. A test interpretter is coded or trained to evaluate these user-defined expressions and perform the desired activities. These tests are usually written external to the testing tool. For example, they may be in text files, spreadsheets, or database tables.

R

Record and Playback: A test technique in which the automation tool records or captures user input for automated replay at some later time. Usually, the automation tool creates an editable test script that reflects the user actions that were recorded.

T

Test Automator:

The test automator develops and maintains successful automation based on the test designs provided by test designers, and the automation tools used to execute the tests. Designing tests generally consumes more time than automating, so test automators can usually support multiple test designers.

Test Design:

A test design generally contains the flow of desired test progression, user or system input, and an expected result where applicable. One might say that a test design is "a test"; and test designs are "the collection of tests".

Test Designer: A test designer generally defines or creates test designs--the tests. They focus on the scenarios for testing the application, and expressing those in test designs. Test Designers are not concerned with automation tool details, languages, or complexities. That is the role of the Test Automator.

Test Framework: The collection of associated tools, technologies, and predefined processes deployed collectively as a system to accomplish test automation.

Test Technique:

The physical methods by which test automation is expressed or captured. The most common of these would be *Record and Playback*, *Functional Scripting*, and *Keyword-Driven* methods.

Tester: See "Test Designer"

This Page Intentionally Blank

Lesson

1

An Ideal Testing Framework

What if we could design the ideal test automation framework? What would that be like? What might we consider "ideal"?

Come in and explore some interesting ideas, and tell us what you like, and don't like.

- An Ideal Test Framework
- Perfect Reflection?
- Butterflies and Happy Cries

An Ideal Test Framework

Test Designs and Test Tools are Separate Entities

Nearly all of the most commonly used automation techniques tie our Test Designs -- the actions or scenarios we wish to perform -- to the test tools that will execute them. In most tools, these actions are forever coded in the cryptic scripting language of the test tool itself!

But our Ideal automation framework will allow us to break that tie. After all, when we go to buy a car, do we describe the car to the dealer in terms only metal-stamping machines can understand? Of course not!

Likewise, Test Designers should not have to express test actions and user scenarios with unfamiliar terms. And they definitely shouldn't have to do it *twice*: once for humans, and again for a test tool. Similarly, Test Automators should not have to become familiar with the subject matter or domain covered by the application. They can focus on test automation and leave the application domain expertise to the Test Designers.

Our ideal automation framework allows us to express tests using our application's vocabulary. This provides some wonderful advantages:

1. Designers don't have to learn the complex testing tools.
2. Design tools are separate from testing tools.
3. Tests can be executed by more than one test tool.
4. Testers can move from project to project effortlessly.

With test designs separated from test tools, what other advantages might we gain?

- ○ Test Design can start early
- ○ Tests can migrate to new tools, or use more than one tool
- ○ All of these are true
- ○ Automators don't have to become domain experts

An Ideal Test Framework (cont'd):

Test Designs Should be Modular and Reusable

A very critical factor in the success or failure of an automation framework is the amount of time spent maintaining it. Whenever something exists more than once, and it breaks, that something must be fixed everywhere it exists. One of the best ways to reduce this maintenance is to eliminate all instances of duplication.

In our Ideal automation framework we want to maximize reusability by allowing Designers to define simple actions, or sets of actions, that can use different parameter values. For example, they can have a single test design that performs login, but can call it with 15 different userids and passwords.

This sounds so simple, but some automation techniques and tools don't do this very well. Imagine if you needed 15 different cars to enable you to travel 15 different speeds. It sounds ridiculous, and it is; but some automation techniques and tools are ridiculous in just that way!

In our Ideal automation framework, remember this: reuse goes beyond the logical, iterative and variable execution scenario mentioned above. It also extends to reusability across testing tools and testing platforms.

Question:
A given suite of tests for an application requires 25 different login attempts. 20 of these should be successful, while 5 of these are expected to fail. Assuming our test designs have been created for maximum reusability and ease of maintenance, how many different test designs will there be to login to this application?

- ○ 25
- ○ 1
- ○ 2

Test Design Vocabulary is User-Defined

We don't want to find ourselves constrained by the limited vocabulary or API of the testing tools we deploy. In fact, we want to express our tests using the vocabulary that is best suited for the application we are testing, or the application user we are trying to mimic.

For example, an accountant using the "PayUs Pro" accounting application wants to do things like "close account". Yet, the human resource rep using "FireThem 2003" wants to be able to "delete employee". These are similar tasks--removing items from a list or database. But the two applications use a different vocabulary for these similar activities.

In our Ideal automation framework, our test designs need to be able to work at that level of abstraction. Our Test Designers need to be able to use whatever vocabulary is appropriate. In fact, for Test Designers, they get to define the vocabulary that will be used!

With that in mind, select the two most truthful statements concerning the impact of a testing tool API on our Test Design vocabulary?

- ○ Test Tool API has no impact on Test Design vocabulary
- ○ Test Design vocabulary will be application-specific.
- ○ Test Design vocabulary should closely match that provided by the underlying test tool API.
- ○ Test Designers must learn the API of the underlying testing tool.

Designs are Easy to Understand and Execute

We have seen some mighty crazy programming languages in our time. Few test automation languages are able to satisfy this important concept: *We do not want to do this twice!*

We do not want to write a test design that is human-readable, and then go and write it again in some diabolic scripting language. That does more than double our efforts, because we now have to keep 2 different assets synchronized; or spend time periodically trying to deduce which one is correct, and which one needs to be updated!

Instead, our Ideal automation framework allows us to use one easy-to-understand test asset that is suitable for both man and machine. A tester, an auditor--heck, even a *developer*--should be able to readily interpret our test designs. And those same test designs are fed to our automation framework, and executed by our testing tools.

Question: Are audits of automated testing greatly simplified by this type of framework?

- ○ Yes
- ○ No

Framework is Application-Independent

There is little worse than spending a great deal of time getting something just right, and then finding out you have to do it all over again. Yet, worse than that, is *knowing* you are spending a great deal of time on something, and *knowing* you will be doing it all over again!

Yet, that is what happens when people build up a testing framework designed to test one target application. They make a bunch of scripts and libraries that exercise this one application very well. But when it comes time to move on to another application--a new testing framework must be created. They must do it all over again.

Our Ideal automation framework recognizes that a finite set of common components is the foundation for an infinite number of unique applications. If the framework can handle these common components well, we will be able to test the vast majority of applications while writing little or no application-specific testing code.

Question: What suggests a test framework is application-independent?

- ○ It is very effective at testing one target application.
- ○ It can be customized to test new applications.
- ○ It can effectively test many different applications.

Journal: Perfect Reflection?

Go through the lesson first. Then tell us if you feel there is something missing from this Ideal automation framework? Any concept or Ideal you tend to disagree with; or that doesn't sit quite right with you?

Can you think of additional benefits this Ideal framework offers either directly or indirectly?

Butterflies and Happy Cries

During an organization meeting on testing across several departments in different divisions, I suddenly realized just how much flexibility our keyword-driven framework provides us, and how we were able to take it for granted.

We were nonchalantly reorganizing test personnel -- moving them in and out of different departments based on need. There was little consideration for whether or not the testers could readily move into a project and decipher the test automation in place. Remarkably, I realized it didn't concern me either.

In using the SAFS Framework, all these departments used the same test development tools -- even if they used different, or augmented underlying toolsets. A tester in one department could readily jump into the framework used in another department because test development was not tied to the tools that would execute the tests!

For some reason the insight just hit me especially profound that day. With a happy grin, I made sure all the managers present were fully aware of the fairly intangible benefit the framework was providing!

We have been using the SAFS Framework since its very beginning -- back when it was called "the DDE", or "the Data-Driven Engine".

Lesson

2

Record and Playback

Record and Playback has been one of the prime features that sells automation tools. Yet complaints about the shortcomings of this technique abound.

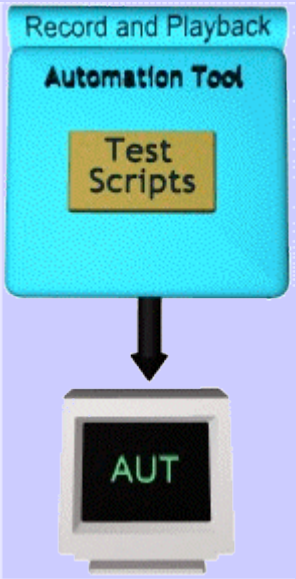
Can Record and Playback really be as bad as people say it is?

Record and Playback Pros and Cons

Antidote or Anecdote?

Record and Playback Pros and Cons

There is some Good in Record and Playback Techniques



Let's face it, there is nothing easier or faster in test automation development than having a tool record user actions. And that can sell a great many automation tools, too! Unfortunately, in real-world test automation practice, recorded scripts never live up to the billed fanfare.

Still, one cannot easily counter the following point about recorded scripts:

- Rapid and somewhat automatic script development

Let's take a *very* quick look at how this works, and then point out the pros and cons of this strategy.

Recording Einstein on Google

Lets pretend we are going to google "Einstein" in News items on Google.com

The following user actions are necessary to do this:

1. Launch Google.com
2. Click the "News" link
3. Enter "Einstein" in the News Search field
4. Click the Search News button

A recorded script will capture all these actions (and every mistake made in performing them). In the end, you should have a recorded script that will perform all 4 of these actions in the desired sequence.

Issues with Einstein on Google.com

Let's quickly note some potential issues with this recorded script:

- The script can ONLY search for "Einstein"
- It can only search in News
- It can only search on Google.com

If we must search for other items, then we must record additional scripts. And these scripts will record all the same user actions again. They will all launch Google.com. They will all click the News Items link, and they will all click the Search News button.

If we ever have to change any one of those activities, such as when "News" becomes "News Items", then we must fix every single script we ever recorded with this activity!

Uses for Recorded Scripts

Yet, amidst all the doom and gloom, there are some scenarios in which this strategy is desirable.

Recorded scripts are extremely useful as easy-to-make "throw away" assets--assets not intended or needed long-term. They are helpful in test tool scripting language evaluations, component object recognition evaluations, and anywhere script maintenance is considered a non-issue.

A great example for this is debugging problems in other scripts. Sometimes the easiest way to evaluate what is "right", is to simply capture the scenario in a recorded script. Then you can analyze the differences between the problem script and the recorded script. (Assuming, of course, the recorded script will actually playback accurately and perform the required actions.)

Problems with Recorded Scripts

As already mentioned, the most noticeable issue with recorded scripts is that they are not very useful for production testing "as is". After we painstakingly capture them, we usually must carefully "fix" them--especially if we were not so painstakingly meticulous when we recorded them.

Recorded scripts have other problems, too:

- Often difficult to read and follow logically
- Much higher maintenance than alternatives
- Require testers with programming experience to develop and maintain
- Recorded in a tool-specific scripting language
- Only used to test one application

Journal: Antidote or Antecdote?

What has been your experience with Record and Playback scripting?
Do you have a novel scenario for using this technique?
Or, do you avoid it like the plague?
Or, perhaps you are wondering where this negativity is coming from?

This Page Intentionally Blank

Lesson

3

Functional Scripting

Functional Scripting is the natural evolution beyond *Record and Playback* into a toolbox of scripts and reusable function libraries.

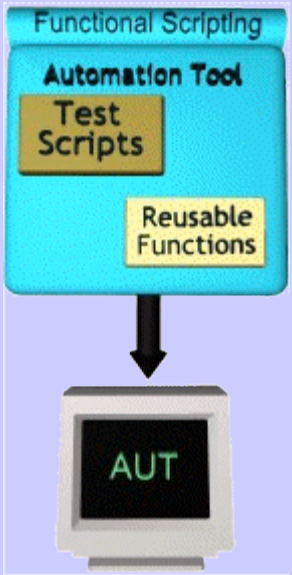
Let's see why this is better than Record and Playback, and ask ourselves, "What could be better still?"

Functional Scripting Pros and Cons

Dollars and Sense?

Functional Scripting Pros and Cons

Fixing the Woes of Einstein on Google



We graduate to "parameterized" scripts, or script libraries, once we realize that Record and Playback is not going to work as well as we had hoped. If we go back to our "Einstein on Google" example in Lesson 2, we can readily see that we need these recorded scripts broken down into functions that can accept parameters.

The functional breakdown of the scripts might result in these library functions:

- LaunchURL (aURL)
- SelectLINK (linkText)
- EnterSearch (searchText)
- SubmitNewsSearch ( )

Advantages of Functional Scripting

We can readily see now, that we have a huge opportunity to reduce maintenance by providing a high degree of functional reuse.

Here is an example of a test script demonstrating this reuse:

```
LaunchURL ("google.com")
SelectLINK ("News")
EnterSearch ("Einstein")
SubmitNewsSearch ( )
EnterSearch ("Newton")
SubmitNewsSearch ( )
EnterSearch ("NOT Wayne Newton")
SubmitNewsSearch ( )
```

Lurking Behind the Facade

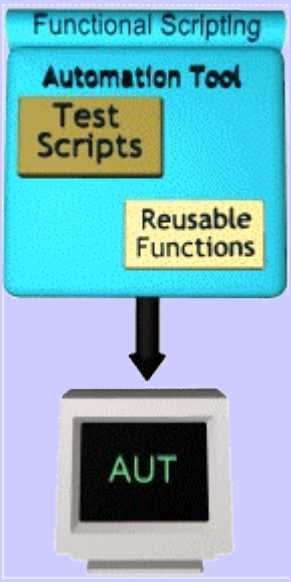
Remember our tests now call various functions like these:

```
LaunchURL ("google.com")
SelectLINK ("News")
EnterSearch ("Einstein")
SubmitNewsSearch ( )
```

What lurks inside these functions, however, is essentially the same thing we had in our recorded scripts. The script has just been broken apart and moved into separate functions for easy reuse. And now the functions take parameters that allow us to do more than just one hardcoded thing.

But they are still test scripts written in the context of the testing tool.

Better, but not Ideal



So, we are much better off than we were with recorded scripts. Yet, we are far from that "ideal" framework we defined in Lesson 1. Some of the issues are not readily obvious given the previous script fragment.

Some concerns about the Functional Scripting strategy:

- Framework is tied to the tested application
- No external support for an app-specific framework
- Requires programmers to maintain the tests
- Still relatively high app-specific maintenance
- Tests are tied to the test tool scripting language
- Tests can be broken by changes in the test tool

Journal: Dollars and Sense?

Hopefully, we all recognize the important advantages Functional Scripting provides over Record and Playback. Still, the added flexibility doesn't come free.

Can you think of other scenarios when Record and Playback might still be preferred? Or, perhaps you might think that if there is no time for Functional Scripting, then there is no time for automation--period? What key aspects of our Ideal automation framework does Functional Scripting not satisfy?

This Page Intentionally Blank

Lesson

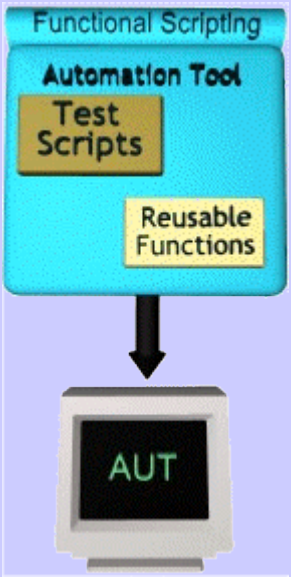
4

Keyword-Driven Testing

Keyword-Driven testing liberates you and your tests from the automation tools that will execute them. Let's see what else it can do that the other techniques cannot offer.

Keyword-Driven Pros and Cons

Removing the Test Script Context



Lets examine the test script from our lesson on Functional Scripting:

LaunchURL ("google.com")
SelectLINK ("News")
EnterSearch ("Einstein")
SubmitNewsSearch ( )

We have, essentially, a list of common user actions that have application-specific meaning. Unfortunately, they are still functions trapped inside the context of a specific testing tool. In addition, the functions themselves do very specific things to components on Google.com, and we will need to correct that.

Removing the Application Context

Remember the concept, "a finite set of common components make up an infinite set of unique applications"? Instead of capturing small pieces of Google-specific funtionality to make up our functional script library, we need to code generic component functions that know how to do simple GUI component tasks *on any application we wish to test*.

Example generic functions:

- StartBrowser (google.com)
- ClickLink (Google, News)
- EnterText (Google, SearchField, Einstein)
- ClickButton (Google, Search News)

When we develop a relatively small number of predefined component functions like this, we provide an easy-to-read syntax that anyone can understand. The syntax is not tied to the testing tool in use. It is not tied to the application we are testing. And we start enabling test portability to other tools and platforms.

Define Tests as User Scenarios

The previous sample of component functions shows what a Test Automator will use to make the automation happen. Test Designers, on the other hand, just focus on defining keywords that convey critical user scenarios that need to be tested.

The Test Designer will define tests using application-specific keywords:

1. LaunchGoogle
2. SearchNews "Einstein"
3. SearchNews "Newton"
4. SearchNews "NOT Wayne Newton"
5. ShutdownGoogle

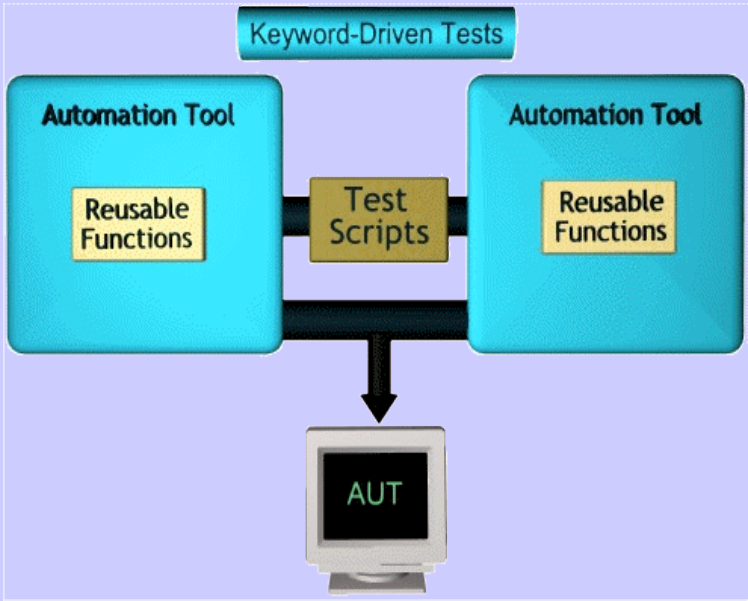
The Test Automator will accomplish these using the available component functions, and all of this is outside the confines of any specific automation tool.

Keyword-Driven Functional Routing

The last step in a fully keyword-driven framework is to provide the functionality that can read the test input--often a text file, spreadsheet, or database--and route the tests to the appropriate low-level component functions for execution. This isn't absolutely required if we are not interested in test portability; but in a world of ever-changing technologies--we are!

The tests can now be developed independent of the test tool that will execute them. The keyword-driven framework must only implement basic component functions in order to test a huge variety of applications. And we can readily migrate to a different tool, or a different platform, simply by reimplementing those same component functions in the new tool environment.

The Keyword-Driven Pros



We have now stripped out all attachment to any specific testing tool, and all proprietary automation APIs. This must have bought us something!

- Test Designs are easy to read, execute, and enhance
- Test Designs are now separate from Test Automation
- Test Designs can be developed by non-programmer testers
- Automation Framework is application-independent
- Automation Framework can be shared across boundaries*
- Automation Framework support available across boundaries*

* Departments, divisions, companies, countries, and continents.

The Keyword-Driven Con

Some things, sometimes, sound too good to be true. Keyword-Driven automation satisfies the vast majority of features we want in our "Ideal" automation framework. But that does not come for free!

The Biggie: The time and effort necessary to develop a keyword-driven framework from the ground up is not insignificant or trivial. (So don't do it. Use SAFS--which just happens to be free! http://safsdev.sourceforge.net)

In addition, there is no automated test script development like that in Record and Playback frameworks. So, it *might* take longer to develop your application tests. (However, this is hotly contested by some experienced test automators.)

Journal: Benefit or Behemoth?

Many people don't take the time or effort to go that extra step towards keyword-driven automation. After all, in the absence of a keyword-driven framework at your disposal, there is additional framework infrastructure that must be created above and beyond your typical automated test scripts.

Based on this lesson, or even on your own personaly experience; share your comments on additional benefits you perceive; or on any problems you fear concerning keyword-driven testing.

Contrasting Functional Scripting and Keyword-Driven Tests

The lesson on Functional Scripting provides some detail and examples that look very similar, or nearly identical, to those used for Keyword-Driven Tests. In fact, some Keyword-Driven examples seem to come full-circle and discuss application-specific tests we were told to avoid in Functional Scripting. What's up with that!

The answers lie in the picky little details.

Functional Scripting requires the full application-specific details of the test be codified and trapped in the scripting language of the automation tool. Sure, we can implement test scripts that look like keyword-driven tests, and that may be all we care to do. But those tests are destined to be executed by one tool, and one tool only. They will not be able to migrate to new environments or tools.

The same Keyword-Driven test has only a small fraction of the code--the generic component functions--written in the language of the automation tool. All the application-specific test designs and scenarios are written externally and independent of the testing tool.

Thus, with a Keyword-Driven framework we can make any number of different testing tools execute these tests. This even allows us to use multiple tools at the same time--taking advantage of each tools individual strengths and features.

Lesson

5

Mix and Match

No one technique can ever stand by itself and solve the world's testing problems. So let's keep a cool head and realize there is a time and place for everything!

All For One, and One For All

All For One and One For All

One Technique is Never Always Right

In Lesson 2 we learned about Record and Playback strategies and the difficulties inherent in them. In Lesson3, we discussed the evolution to Functional Scripting that ocurred to mitigate these issues. Then, in Lesson 4 we took that next bold step--Keyword-Driven Test Automation.

As we've seen in these lessons, there is a time and place for each one. Different circumstances, different needs, and far-reaching decisions from managers that don't know what they're talking about can affect the choice of the primary automation strategy.

If you have access to a Keyword-Driven framework, you should devote most of your effort in using it. However, there will still be times when Functional Scripting, and even Record and Playback might be viable options.

Evaluate Risks, Costs, and Choose Wisely

The fact is: we need to evaluate the task at hand, the automation strategies available, the risks and costs of ongoing maintenance, and make decisions based on these and other factors.

Even if you are blessed, and have the ultimate keyword-driven automation framework available to you; there are times when something as lowly as a simple recorded script is the right choice. On the other hand, a functional script may be the ideal answer for an expected 10,000 iterations of adding new accounts with 10,000 variations of "Bob and Doug McKenzie".

And believe it or not, sometimes the right strategy for a particular task is not to automate. In those cases, test automation is the right road to be less travelled.