testudio automationlab

FUNCTIONAL TESTING

The purpose of this white paper is to give you a somewhat detailed overview of functional tests' design on **TeStudio** Automation Lab. You will be presented some of the tool's main concepts regarding its test's automation approach. After reading this paper you should get a clear idea of the tests' anatomy on Automation Lab and also get some insights about how can these concepts be transposed and applied to your own reality.

A recommended reading to better contextualize this white paper is Telbit **TeStudio** flyer which provides a high level **TeStudio** suite overview including the module presented here in greater detail – Automation Lab.

SYSTEM UNDER TEST

For the sake of simplicity, in this white paper we'll use a hypothetical classical three tier application composed by a RDBMS backend, an application server and a GUI front-end. Let's name this application SUT.

Testing SUT will allow us to demonstrate quite a few testing patterns easily applicable to other scenarios. Here we will follow a top-down approach, since it allows us to, first of all express the more general subjects and after that, to drill down to the core components of the test.

TEST PURPOSE

One of the requirements of SUT – like so many other applications – is to authenticate a user given its login and password. Testing this requirement, will be the main goal of our tests. We will create two tests: the first one, to test a valid login and the other one, to test an invalid login attempt.

Since Tests on Automation Lab are built composing one or more previously created steps, we'll start by creating these, and after that associate them with each test in a proper way.

Minor details will be trimmed from the test steps' definition due to their lack of relevancy in this context.

STEPS CREATION

Step 1: Database Setup – a SQL script whose purpose is to put the backend database in a known state. Its content is nothing more than a bunch of deletes or truncates and a couple of inserts to fill some required look up table values.

Step 2: Create User – is also a SQL script whose purpose is to create and insert a valid user in the proper table(s).

Step 3: Monitor Application Server Logs – a bash script which will be executed in a remote host via a SSH connection. It will run a tail command on a text log file to get feedback regarding the application server's operations. This kind of steps has a crucial usefulness for later debugging purposes when something goes wrong with the system under test.

Step 4: Do Login – a GUI automation script recorded by Automation Lab. Since this script is specified in a custom DSL, we'll present here a brief sample:

```
navigate <WEB_APP_URI>
write "jdoe" (name="login")
write "jdoespass" (name="password")
click button (value="submit")
capture page
log
```

The above script has five different types of instructions:

1. **`navigate`** – opens the target browser and points it to the target application URI. This URI will be translated according to the active Environment (more on this subject below)
2. **`write`** – emulates a set of keystrokes in a GUI widget (e.g. a text box)
3. **`click`** – emulates a mouse left click in a GUI widget (in this case, a button)
4. **`capture page`** – captures the full page (scrolling it to the bottom if necessary) as an image to attach to the test execution evidence
5. **`log`** – dumps a textual representation of the html document currently active on the target browser. As you'll see bellow, the output of this instruction will be used to auto-evaluate the step result.

Step 5: Check User Access Tables – another SQL script which makes a select on a table where are inserted all authentication attempts.

Test Environments

Each of the above mentioned steps has a named alias. These aliases will allow Automation Lab to map them to their effective target hosts on the execution stage. Target hosts are configured in an Environment and as appropriate, tester is asked for IPs, credentials, connection types, etc.

Untying the steps' executable content from the physical infrastructure's definition turns possible to setup several test Environments (replicating what usually happens on a test department) while keeping the entire test set ready to be executed in any of them according to the project demands.

Test a successful login attempt

Having created those five steps, all we have to do now to complete the first test is to drag them to our test record to set up their workflow which will be:

Step 1 is the test starting step

- Step 2 starts after Step 1 ending
- Step 3 starts after Step 2 ending and continues to run until the end of the test
- Step 4 starts after Step 2 ending
- Step 5 starts after Step 4 ending

Since, at this point we have not – yet – defined any kind of test oracle for our steps, by now the execution of this Test will be driven by Automation Lab – in a wizard like interface – but the evaluation of each Step and hence that, the overall test result will be delegated to the tester. Later – on section Steps evaluation – we'll show how to define a test oracle for each Step to turn this test a fully automated one.

When running these steps on a test case context, all their output (be it text or captured screen shots) and the full test environment characterization are collected and attached to the Test Execution Log.

Test a failed login attempt

To design this test, all we have to do is, duplicate the previous one, remove the second step and revise its designation/description. All the remaining workflow (giving that Steps 3 and 4 are updated to start after Step 1 instead of the removed Step 2) is still valid and suitable enough to test our failed login attempt since we removed the step which creates the user on the back-end database. This way, when executing Step 4 on this context, the login attempt must fail.

This – even quite simple – scenario makes a good job illustrating the point of steps (re)usage on Automation Lab. Our field experience shows that on test projects containing more than one thousand test cases, on average, each step is used on fifty test cases.

STEPS EVALUATION

In the above sections we described two tests and their composing steps. As we mentioned there, their execution is automated by Automation Lab but the evaluation result is still fully delegated on the tester. In this section we'll show how to automate each step's evaluation defining a test oracle to each of them.

On Automation Lab, a test oracle specification is created using regular expressions. To be fully automated, a test must have at least one step with its test oracle defined. Test oracles are composed by three properties: a timeout value (in seconds, with a mandatory value greater than zero), a fail and/or a pass condition. From now on, we'll name steps with a test oracle defined, automated steps.

Test oracles have the following semantics:

TEST ORACLE SETUP		AUTOMATED STEP RESULT		
Fail Condition	Pass Condition	Fail condition matched	Pass condition matched	Timeout expires before a match
Defined	Not Defined	Not OK	N/A	OK
Not Defined	Defined	N/A	OK	Not OK
Defined	Defined	Not OK	OK	Aborted

Test evaluation semantics

Tests on Automation Lab have an execution status property which may be given one of the following values: OK, Not OK, Aborted or No Run.

A test is given an OK value if, and only if, all their automated steps are evaluated as being OK. When at least one step is evaluated as Not OK, then the test is also evaluated as Not OK. If at least one automated step is evaluated as Aborted and none of them is evaluated as Not OK, then the test is evaluated as Aborted. This last test execution status – Aborted – usually happens when there is something wrong on the test environment or in the steps definition.

Defining Test Oracles

Now we're ready to define the test oracles to the steps belonging to the first test:

STEP	FAIL CONDITION	PASS CONDITION	TIMEOUT secs
Step 1 Database Setup	`ORA-\d{3,5}`	`\d+\srows\saffected`	10
Step 2 Create User		`1\srow\sinserted`	5
Step 3 Monitor App Server Logs	`Exception|ERROR`	`jdoe\slogs\sin`	60
Step 4 Do Login		`Wellcome\sjdoe`	15
Step 5 Check User Access Tables	`ORA-\d{3,5}`	`1\srow\sreturned`	10

As you can guess from the above table, fail conditions mostly catch execution errors (on Steps 1 and 5 the expression catches the most common types of Oracle PL/SQL error codes) while pass conditions mostly assert the expected steps' side effects.

Unlike this proposed sample, where all conditions are a bit loose, both condition types can - and in fact they should - be as "tight" as possible to prevent errors from remaining undetected.

Given Automation Lab's concerns on knowledge gathering regarding the system under test, along the test project's lifecycle, test oracles tend to become a lot narrower. This way, chances are that even if a defect leaked to a production release, when retesting – revising testing oracles to catch that flaw – is it much more unlikely that similar defects remain undetected on following releases.

As steps' reuse rate increases, chances are that when revising their test oracles, testers are at the same time improving the effectiveness of several tests and due to that, also increasing their associated requirements testing wideness.

We should also note here that, purposely, the last two steps do exactly the same functional validation, however each one does it on a distinct system layer – after a successful login attempt, Step 4 asserts the expected visual output on the GUI front-end, while Step 5 asserts that there were created the appropriate records on the back-end database. Automation Lab encourages an "all layer probing approach" since this way, chances of catching software faults are dramatically maximized. This double-checking does not raise test set execution costs since both their execution and evaluation are fully automated.

CONCLUSIONS

This white paper presented a brief and skimmed technical overview of Automation Lab's approach to functional test a software system.

Not with standing having much to more to say about designing and automating tests, we hope this paper showed Automation Lab's suitability to design, execute and evaluate software functional tests and also that, its approach is pretty much aligned with Testing/QA Departments mission.