December, 2008

testing experience

The Magazine for Professional Testers

printed in Germany

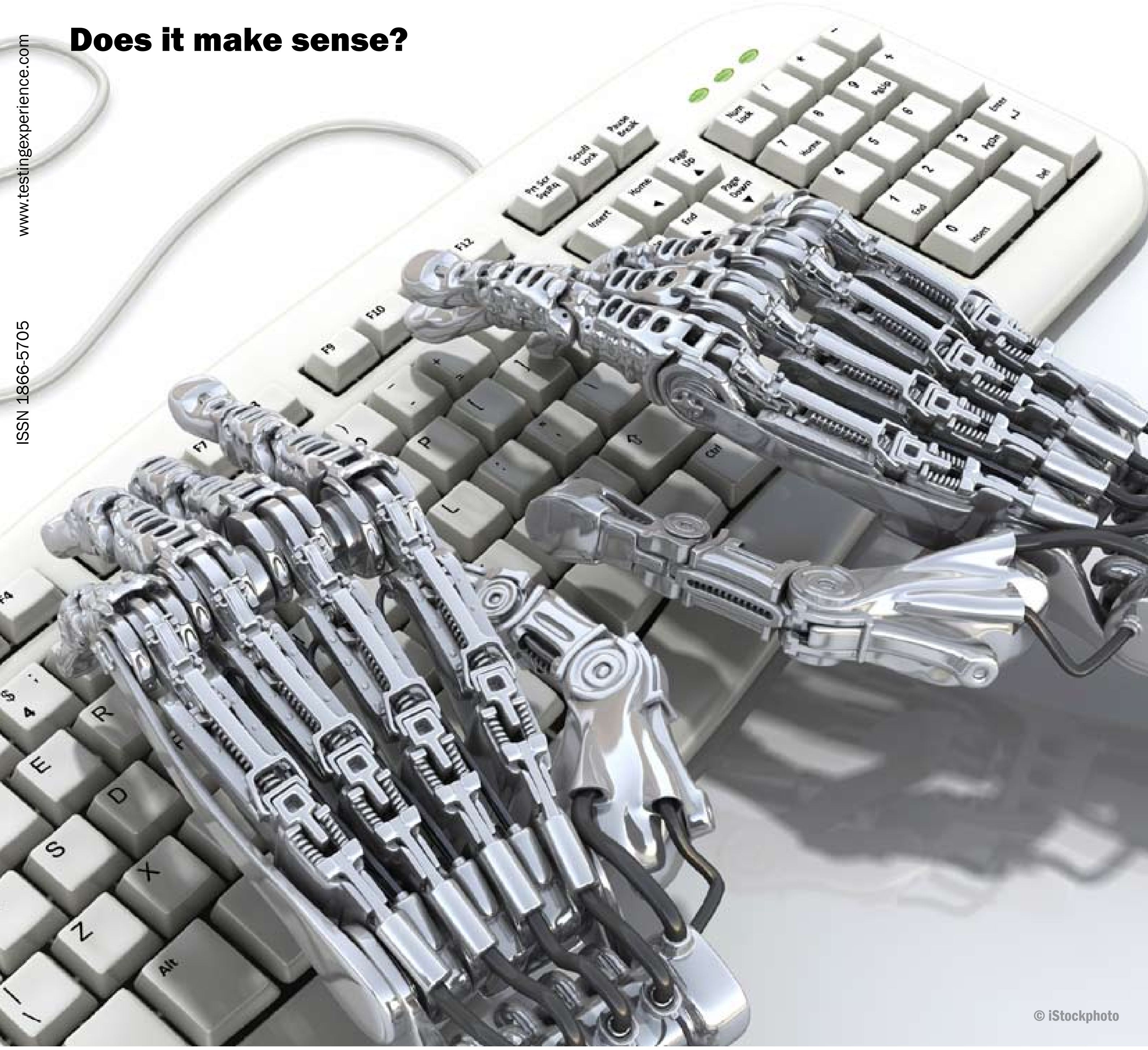
free exemplar

www.testingexperience.com

ISSN 1866-5705

Test Automation -

Does it make sense?



The Record & Playback Fairy Tale

by Koen Wellens

"Once upon a time, there was this nice little tool that could automate test cases with a simple record and playback action." It could be the beginning of a wonderful fairy tale, but not a realistic one. Test automation is more than just record & playback. It is an entire process with its own challenges: aligning people with different points of view, avoiding common pitfalls and creating a solid structured approach. Let us take a closer look at the path of successful test automation.

Record & Playback

What does "Record & Playback" mean in the world of test automation? A test tool captures user actions in record mode, such as a mouse click or a keyboard stroke. When the recording is stopped, the resulting script contains several steps to simulate the captured user actions. If you click on a "Play" or "Run" button, the tool executes the script and all your recorded user actions will replay automatically. This will probably work fine during the first execution, but what if the script is played against a new release or an error occurs? Moreover, you can certainly not call this a test, as you did not include any validation at all!

Let us take a closer look by using an example. The application under test is a flight reservation system. The following tests need to be automated:

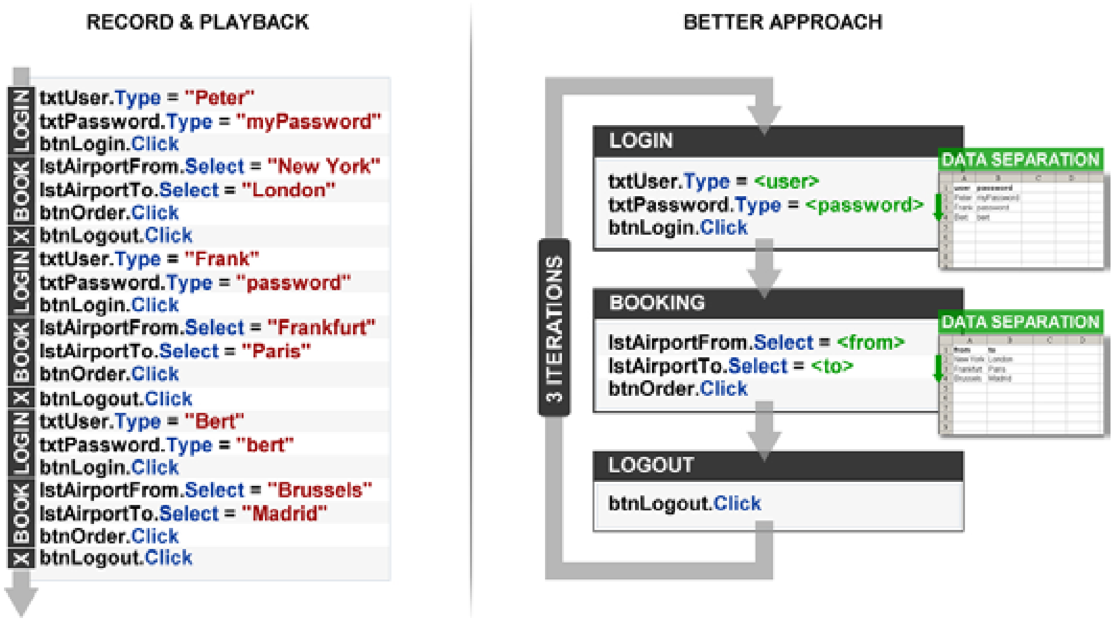
- Log on using 3 different accounts;
- Book 2 flights with different airports for each login.

Record & Playback approach

Press the record button, login manually, book 2 flights, logout and authenticate again using the second login. Book another 2 flights. Perform this action once more and stop recording. Now you have automated the given test cases.

A better approach

Press the record button, login manually, book 2 flights and logout. Stop the record mode. Now you will iterate the recorded steps programmatically by using a loop. Next, replace the static data in the steps (user name, password and selected airports) by a dynamic value. A dynamic value is a value retrieved from e.g. an external data pool (spreadsheet, database, flat file) at run-time. Finally add steps manually to verify that you have logged in successfully and that the correct ticket price is displayed.
Now your script is automated to run the same tests, but with 1/3rd of the code compared to the Record & Playback approach. Additionally, you have added validation and separated data from the script (called "data-driven test-



```
txtUser.Type = "Peter"
txtPassword.Type = "myPassword"
btnLogin.Click
lstAirportFrom.Select = "New York"
lstAirportTo.Select = "London"
btnOrder.Click
btnLogout.Click
txtUser.Type = "Frank"
txtPassword.Type = "password"
btnLogin.Click
lstAirportFrom.Select = "Frankfurt"
lstAirportTo.Select = "Paris"
btnOrder.Click
btnLogout.Click
txtUser.Type = "Bert"
txtPassword.Type = "bert"
btnLogin.Click
lstAirportFrom.Select = "Brussels"
lstAirportTo.Select = "Madrid"
btnOrder.Click
btnLogout.Click
```

```
LOGIN
txtUser.Type = <user>
txtPassword.Type = <password>
btnLogin.Click

BOOKING
lstAirportFrom.Select = <from>
lstAirportTo.Select = <to>
btnOrder.Click

LOGOUT
btnLogout.Click
```

ing"). It is obvious that the second approach is easier on maintenance, more robust and last but not least: it has finally become a real test by adding validation.

Different people, different perspectives

The term "record & playback tool" is misleading as you can do a lot more with these tools. You can add validation, separate data, connect with a database, etc. The term leads to a troubled view on test automation. A tool vendor tends to stress the ease of use of the record & playback capabilities, a manager looks at test automation as a simple record & playback task, lowering resources and time allocation to set up automated tests. A test automation engineer will soon learn that record & playback is not enough, requesting more time and resources for the automation process.

Estimations of workload and time allocation will change during the process. In practice, there are two common mistakes:

1. **A test automation tool is an extra tester**

A common mistake is that test automation replaces the manual tester by an automated version, the virtual tester. This is not the goal of test automation. The real ROI is the quality improvement of your application through the increase of test coverage, avoiding human errors and speeding up test execution. Manual testers can focus more on the validation of new functionalities instead of spending time on regression tests.

A test automation project requires permanent resources. The projects need to be set up and maintained. A normal process is that the automated regression test set keeps on growing throughout the project life cycle. After all, tests for new functionality become regression tests after the release. These tests can be automated for the next release.

2. **Automate everything**

Another mistake made using test automation tools is that people will try to automate as much as possible. However, not every test is suited for test automation. Regression and other iterative tests are suited for automation, whereas non-iterative tests are not.

Automating everything is counter-productive. After a while, the test automation engineer will have so much maintenance on the existing scripts that the return on investment (ROI) vanishes. You do not benefit from automating non-iterative tests. The code cannot be reused, but still needs to be maintained at each release. If you have a complex test that needs to be executed once for each release, it is not worth automating. The time to automate and maintain this test will always be longer than the manual execution of it. As it is a complex test, you need to create exception-handling code to uncover possible errors.

After a few releases, the scripts become non-maintainable and the automation project is stopped as it has resulted in a bad ROI. If the project gets a second chance, the automation engineer will start all over again, defining the correct scope, thoroughly analyzing the automation project, setting up a structure (framework) and using the experience gained.

Five steps for successful test automation

Many test automation projects do not have a high success rate after one year. Why is that? There are many potential pitfalls:

- The ROI is not tangible. How do you measure the increase in quality?
- Miscalculation in work effort: record & playback appears to be harder than expected
- Non-structured approach resulting in a chaotic and non-maintainable automation project
- No scope is defined: do not automate everything!
- Lack of tool training and experience.

Test automation should be done systematically in a structured approach. There are five steps for successful test automation:

- Planning
- Preparation
- Proof of concept
- Implementation
- Maintenance

Planning

Each phase in the project must be planned together with the project manager, test manager, tester, automation engineers and possibly a steering committee.

You should have a clear view on the current testing methods and information of the infrastructure. Gather information by interviewing key people who have experience in the field of test automation.

Preparation

During the preparation phase, you should define a pilot project and select the test cases that need to be automated. Define the roles and responsibilities and prepare the input data (data management).

Proof of Concept

The test automation tool must be configured in order to be compatible with the application under test. The tool must be able to capture user actions. Since more and more application types are being used, the configuration part is not as easy as it seems. Application types can be Web, ActiveX, .Net, Java, SAP, Siebel, …

It is possible that the test tool will not recognize the application "out-of-the-box", which means you need to configure the tool, server or even the application before user actions are recorded.

Next, you need to automate a limited amount of test cases to prove that the application can be automated. Select an easy, a normal and a complex test case. Demonstrate the execution and reporting of the automated test cases to the decision makers.

Implementation

If the proof of concept is successful, the automation of the selected test cases (scope defined at preparation step) can start. Analyze the selected tests, think about data separation, functional decomposition (separate your code into reusable functions), reusability of certain business components.

Modularize your script into clear-cut building blocks. Finally, you can press the Record button and put method into practice.

During this phase, you can set up a test automation framework. In brief, frameworks can vary from documentation on tool usage (naming conventions, standard approach) to a full-scaled framework based on a spreadsheet or database layer. These complex frameworks are often project-dependent and require a lot of effort at the start of the automation project. Documentation on standardization is an absolute necessity. A layer-framework can be useful, depending on the project. In general, tests are stored in the layer part (e.g. spreadsheet) and can be maintained by manual testers. These tests are written in keywords, making a clear overview of what needs to be tested. The manual testers can even prepare the automated test before a new application release, based on the release notes. The keywords of the layer part are interpreted in the test automation tool by a script engine. The downside is that this engine is very complex. If structural changes are required, the complexity of the script can increase maintenance time.

Maintenance

A test automation project expands together with the application under test. A new release may offer new functionality that needs to be automated. The existing automated scripts need to be maintained, new automated tests need to be added and other persons will take over the automation project in the future.

You have to prepare for these events by providing documentation for each script, by constantly allocating the proper resources for the project and by training newcomers in the tool, the script and the execution.

Happily ever after

Most fairy tales end with the phrase 'they lived happily ever after'. Whether your test automation story has a good ending or not, depends on certain criteria.

Create awareness amongst all players.

Misleading terms like "Record & Playback" soon lead to underestimation of workload and preparation time.

Start with a structured and phased approach.

When there is a solid basis, maintenance becomes manageable and you will benefit in the long term. After all, a test automation process has the same life cycle as the application under test. So get it right from the beginning!

Start by planning the project.

Gather information, manage your data and, last but not least, make the right selection of test cases. A proof of concept can evaluate if test automation is possible on the project. After a positive evaluation, the implementation phase can begin.

Structure your scripts

Modularize and separate data, decompose code into functions and document everything.

The maintenance phase determines the good ending of the automation story. Will the scripts prove to be robust against new releases and will maintenance be manageable?

I hope that your automation story ends happily ever after!