December, 2008

printed in Germany

testing experience

The Magazine for Professional Testers

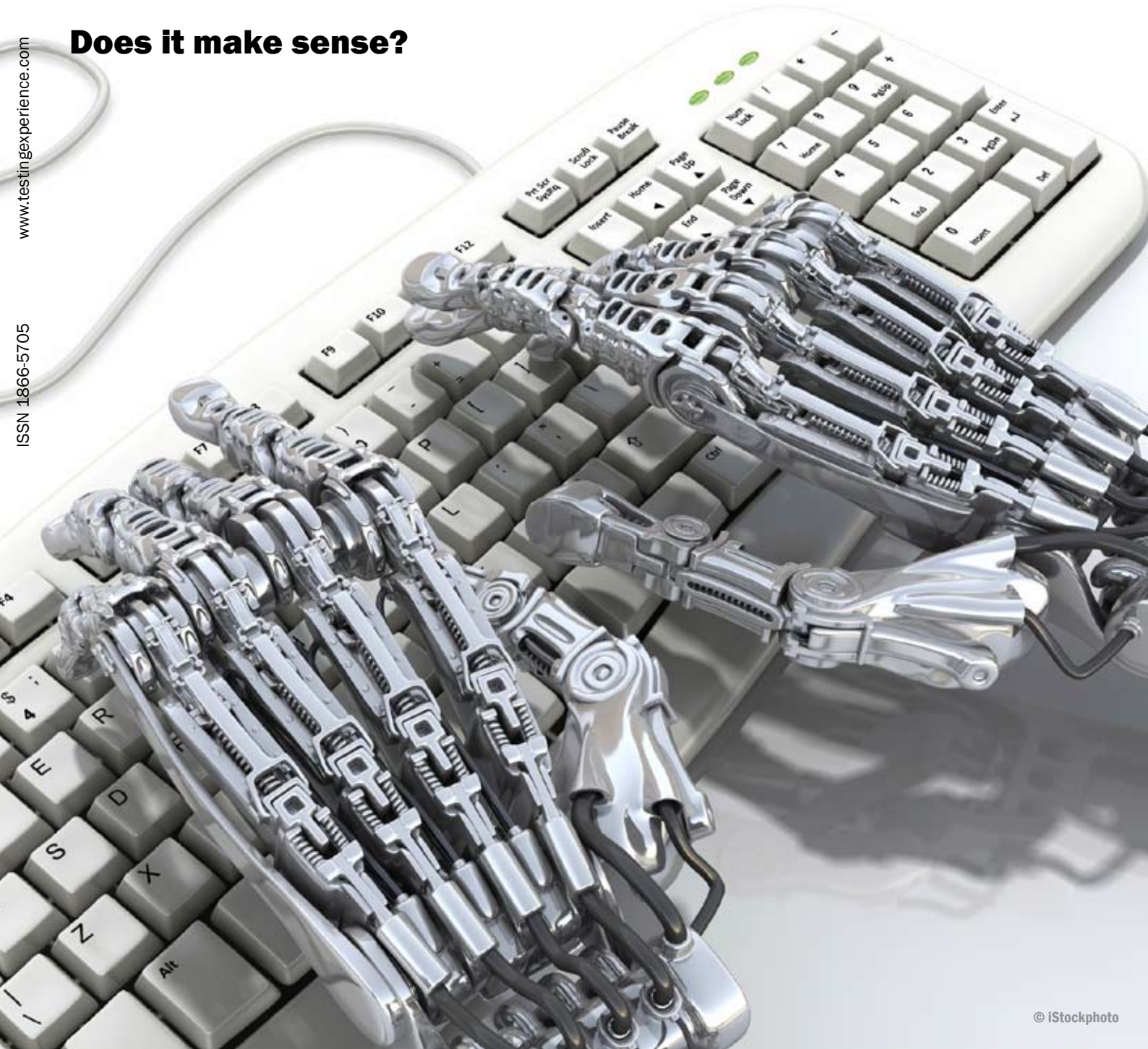
free exemplar

www.testingexperience.com

ISSN 1866-5705

Test Automation -

Does it make sense?



Risk investment Test Automation?

by Konrad Schlude

Abstract:
Test automation can be regarded as a risk investment. Some authors report on a reduction in manual test effort of 80%. There is, however, the number of about 63% of failed test automation projects. On one hand, there is a high potential gain, on the other hand there is a high probability of loss of investment. In order to find out why there is such a gap, we will consider two test automation projects within one company. Whereas one of these projects is considered as a success story, the other was just an investment of several man years without any contribution to the overall software development project. As these examples will show, the success depends on many factors, especially on a good setup.

Obviously, tools are needed for test automation. And every producer of such a tool will argue that his tool is a good one. The question of which tool to use is an important one in test automation, since every tool has its advantages and disadvantages. However, the tool question is not the only question in test automation. It is just one of the reasons why many automation projects fail. Brian LeSuer puts the number of failed automation projects at 63%.

To illustrate some reasons why test automation projects can fail, I would like to give two examples of software development projects. In both projects, test automation was initialized. In one these projects, test automation was and still is regarded as an success, since and there was an important contribution to the software development over several years, In the other project, test automation was nothing but an investment of several man years without any remarkable outcome. Interestingly, both projects were in the same company, in the same building, and on the same floor.

Let's call these projects Project A and Project B. Although there are many similarities, there are also many differences. In Project A, test automation emerged from the test team. The test team developed a strategy of what to automate, so knowledge about the application was available for test automation. In Project B, an external automation team was brought into the software development project, and this automation team acted as a closed group, seperated from the other teams. They did something, but outside the automation team nobody knew what was going on.

In Project A, the test team checked every run of the test robot. With their knowledge, the team members were able to analyze every outcome of every run. If a modification of a script was necessary, a corresponding order of what to modify was generated. On the other hand, when a bug was detected, it was reported immediately to the development team with the request that the bug should be fixed as soon as possible. And the developers fixed the bug as they were told to do. So test automation kept running and kept finding bugs.

In Project B, the automation team was not able to do anything similar, since it was separated from the knowledge of the other teams. Failed test cases were just red marks in the report, and no further analysis could be made.

From the start, maintenance of the test automation was an important topic in Project A. There was a concept of how to react to changes of the application, i.e., the scripts were organized in a modular way. Every GUI element (e.g., button, link, entry field etc.) was identified by a variable stored in a single repository. Furthermore, the team members developed programming guidelines for generating scripts. With this, the amount of time needed for maintenance was quite small.

Again, Project B was different. The automation team just used the capture functionality of the tool without any idea of modularization of the scripts. In the beginning there was very fast progress. After a while, however, they were not able to react anymore to the changes in the application. It was the typical situation where a small change in the application (e.g. button was renamed) led to the modification of an enormous number of test cases. The automation team was busy all the time, but they could not follow the development progress. In order to solve special tasks in some test cases, the automation team programmed scripts. But since there was no concept behind these scripts (i.e., there were no programming guidelines), no one was able to understand the scripts after a short while.

It is not surprising that the automation within Project B failed. Virtually everything went wrong from the beginning. On the other hand, test automation in Project A also had a possibility of failure. For instance, the help from the developers was crucial for success, but there was a lot of goodwill, even though this had not been defined before. Since the personal communication within the triangle of test team, automation team and the developers was OK, every problem could be managed. One reason for the success in Project A was that the involved people brought their knowledge together.

This leads to the following conclusion. Test automation is not a risk investment; it is just a waste of time and money if you don't have a good setup. Just doing "something" with a tool is not a good idea. However, if you define the right frame and bring knowledge and people together, then you have a good chance to substantially improve software development.

[1] Brian LeSuer: Supporting Steps for Successful Test Automation Projects, SQGNE, 2004-05 Presentations

[2] The notion of programming guidelines might be surprising in this context. However, typically manual editing of the scripts is needed although powerful capture-replay tools are used. Within one automation project the amount of manual editing was 80%, and only 20% of the time were used for capturing. And since there is a lot of editing / programming, there should be programming guidelines