Change Detection System for the Maintenance of Automated Testing

Miroslav Bures

Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic
buresm3@fel.cvut.cz

Abstract. Maintenance of automated test scripts is one of the important factors leading to success and return of investment of the test automation projects. As the key prerequisite to maintain the scripts correctly, we need to be informed about all relevant changes in the project scope, specification or system under test to be able to reflect them in updates of the test scripts. In this paper we introduce a concept on how to track these changes in semi-automated way with acceptable additional effort. The proposed solution is based on automated detection of changes in test management and requirement management tools, combined with optional manual inputs and automated scanning of changes in system under test user interface.

Keywords: Test Automation, Automated Detection of Change, Maintenance of Test Scripts, Better Efficiency, Traceability, Change Management.

1 INTRODUCTION

Automated testing represents a promising concept, how to organize the software testing more efficiently. By automation of repetitive, manual (and often intellectually uninteresting) tests, we can invest human effort better in testing of advanced combinations, risk-based testing or error-guessing techniques to generally increase the confidence, the system is tested well. Why the potential of automated testing is used only partially today?

One of the major issues is expensive and inaccurate maintenance of the created test scripts, when the system under test (let us use abbreviation SUT further on) is changing. This is a frequently reported issue, for example [1] or [2].

Also a recent survey we have performed in the Czech Republic among 28 professionals in test automation (currently going to be published), confirms, that the maintenance of the automated test scripts is the most important issue influencing the return of investment (let us use abbreviation ROI further on).

This issue deserves our attention. By enhancing and optimizing of test automation methods, we make the ROI of this technology more probable and earlier. Process of maintenance of the automated test scripts starts with the identification of the scripts,

which have to be updated, when the SUT changes. A simplified view on this process is outlined in Fig.1.

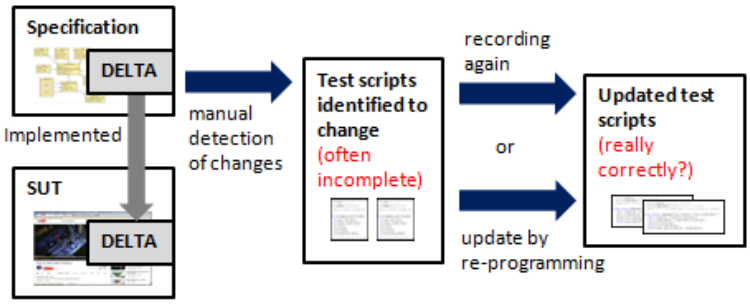

Fig. 1. Simplified process of identification of automated test scripts to update.

When prepared manually with insufficient documentation of the changes in source specification or system under test, the set of test scripts identified to update can be incomplete. It can also contain scripts, which are not needed to be updated. This results in overhead and possible wrong update of the scripts.

2 RELATED WORK

Maintenance of the test scripts has been addressed in literature by various approaches, in which we can basically identify two major streams. The first principal approach is based on a modelling work, where a system organizing automated tests to particular structure or taxonomy is presented, for instance [3].
Although well structured organization of the test scripts can increase the efficiency, a general problem of the proposed models is that they are often not sufficiently flexible to fully adapt to real projects. Various states of test scripts organization and also technological limits on various diverse projects are often challenging the pure modelling-based proposals.

The second approach is based on a traceability often combined with a structuring framework. The idea of traceability is essential here. Generally the maintenance is easier, when we capture and maintain the links connecting the requirements, parts of design documentation, features of the implemented SUT and mainly the respective test scripts. Further in the text, we will be referring to this traceability system. For our purpose we have adopted and extended a generally accepted standard BDTM (Business Driven Test Management) [4] from TMap Next methodology [5]. In the previous work we can find also examples of traceability focused on structuring of the test scripts, for example [6].

The topic of change management has been also a subject of many studies, for instance [7], nevertheless a combination of change management and detection system with a system for the maintenance of test scripts is a topic not sufficiently covered in literature. We approach this key area as suitable subject of our research.

3 PROPOSED SOLUTION

Our solution focuses on two areas, where we can enhance a process of identification of the scripts to be updated: (1) tracking the changes systematically to provide the test script developers with all relevant information for the update and (2) detect changes in the system under test automatically to give the test script developers additional information which can be used in the identification of the scripts. For these both tracks, we are using a model of traceability, which we explain later on.

Let us start with the first part, the systematic tracking of the changes in the specification. In accordance with our experience on usual software development projects, testing sub-team preparing the test scripts is not usually systematically informed about all relevant changes in the scope by a design team or by the project manager. For this reason, we propose a system to help all involved parts to reduce an overhead with collecting the relevant information and increase its accuracy.

The second part, automated detection of changes in the user interface, combines with the previous part. The aim is to have another source of reliable information on what has been changed in the SUT. In this stage, we limit our proposal to automated testing based on the simulation of user's actions in the front-end part of the SUT. With this focus, changes of SUT front-end bring essential information we need for update of the automated test scripts (which are using location of front-end elements to exercise test steps by them). Further on in this paper let us elaborate the first part, the systematic tracking of changes. The second part is currently in a stage of the conceptual design with aim to employ already existing solutions, if any of them will suit our purpose.

4 MODEL OF CHANGE TRACEABILITY

As we have introduced the idea of our enhancement, let us explain the basis on which the solution is based on. The first part of the model (Fig. 2) depicts mutual relations between specification, its changes, a part of changes that is known to test designers, the system under test, test scripts and their coverage and regression caused by changes (defects introduced to the other part of the system by implementing the changes).

By DELTA we mean the change to be implemented. If the project organizes its change management properly, all DELTAs should be exactly specified by formal change requests. Nevertheless, the situation on many projects differs from this ideal and we have to find a solution which can deal with such a situation efficiently.

In our proposal we further focus on the following parts of the model: A set Complete delta (**CD**) represents all changes in the specification (formalized or existing in the minds of the software development team) which arises on the project in a current iteration. A common situation is, that the test design (or test maintenance) team has information only about the part of these changes, a set **KD** in the model. The aim of our proposal is to make the sets **CD** and **KD** equal (or much more the same) with a minimal effort.

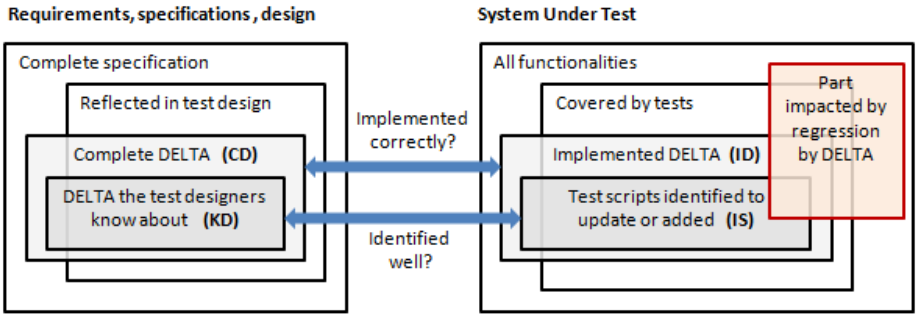

Fig. 2. Basic components of the conceptual model of the change traceability.

Then, the changes should be propagated to the System under test by the development sub-team (a set **ID**) and also to the updated automated test scripts by the test design team (a set **IS**). The reduction of the difference between **CD** and **KD** then significantly contributes to the accuracy of update of the **IS**. And together with this, we need to support the update process also from another line: a traceability of the changes to automated test scripts.

As we have mentioned already, for the test management purposes, it is generally very productive to organize the information on the project to be able to track, which requirements are designed by which part of design documentation, implemented by which features of the system under test and, mainly, covered by which test scripts. We have adopted this general concept, described for instance in BDTM [4] further for change identification and propagating the changes to automated test scripts. The principle is depicted in the Fig. 3.

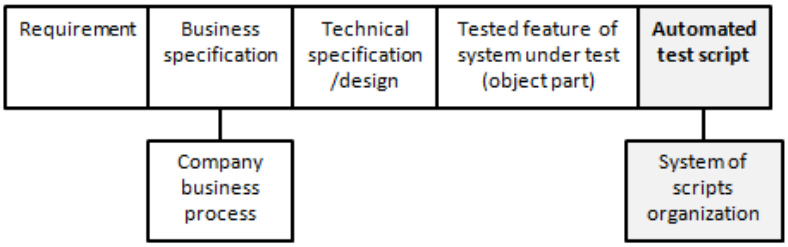

Fig. 3. The model of the change traceability: tracking metadata part.

In addition to the BDTM, we emphasize more explicitly links to company business processes, which help us to work with this source of changes and allows us to align more clearly and exactly with the business part of the organization. Moreover, we have added the system of script organization, which represents taxonomy, name conventions, and physical organization of the automated test scripts, including optional organization of the test scripts by their reusable parts.

5 IMPLEMENTATION AND EXPERIMENTS

In the current stage of our solution, we are handling three possible sources of changes in the project:

(1) Requirements possibly stored in a test management tool. Current tools are offering functionalities for storing the requirements. We are detecting a change by loading the set of requirements several times as the project proceeds and are comparing them. The second possibility is a simple import of information on requirement change from the tool.

(2) Requirements stored in a separate requirement management tool, varying from a simple issue tracking tools, which can be used to capture and formalize the requirements, for example Redmine, to a specialized modelling tools, for example Sparx Enterprise Architect. Here, a process of change detection and import is the same as in the previous case.

(3) Changes detected directly by members of the test automation team. This is an important source of information, which has to be processed. The test designers can add and update this information in the traceability tree by a special front-end application.

An overview of the architecture is introduced in Fig. 4. The system is based on a shared repository, the Traceability data storage, whose core is the traceability tree, the model described in the previous chapter.

For the variants (1) and (2), we gather the data by specialized connectors, which are adapted to the particular tools. A side effect of the solution is a possible consistency check between the various systems storing the requirements, if there are more of them co-existing on the project. Then, the data aggregator and analyzer transform the data to the traceability structure.

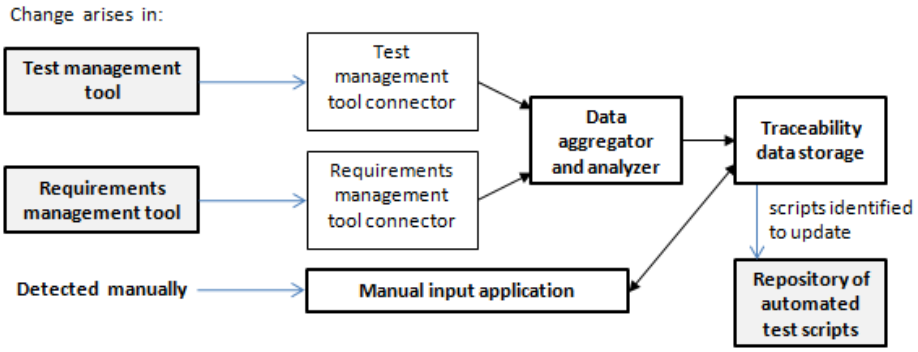

Fig. 4. Conceptual schema of the solution.

Currently, the system is in detailed design and implementation. The conceptual parts have been verified by simulation on real data in a SW development company, where a sufficient set of automated test scripts is available. We have compared a time needed to identify the changes to update the automated test scripts manually and its

possible accuracy with a thorough detailed simulation, testing how this overhead and accuracy would be changed, when using the proposed concept. In this particular case, taking all initial overhead in account, time to identification would be reduced by 40% one year after roll-out of the proposed solution (in the situation that the test designers are really detecting the changes, not simply ignoring them). Then, changes not reflected in test design by lack of information or human mistake would be reduced by 85% in the same period, which is significant. This result encourages us to continue with implementation to get feedback from more experiments, despite the fact that in productive run we expect this result being lower due to overhead which can arise.

6 CONCLUSIONS AND FUTURE WORK

Our proposal aims to increase efficiency of automated testing by focusing on its key issue negatively influencing it: maintenance of the automated test scripts, particularly on its sub-problem, detection of change which we need to propagate in automated test scripts maintenance.

Our approach is based on automated detection of changes form design or test management tools, where the relevant information is stored, in combination with the support for test scripts maintenance team to propagate other information they acquire during the project meetings or communication on general. The main idea behind this concept is to aggregate possible all sources of information available and perform that with minimal effort and overhead.

The results of the first simulation on real data and also an initial feedback from various test managers and engineers in the Czech Republic leads to the conclusion that the proposed concept deserves further evolution and application on the several test automation project to refine and optimize the proposal in the upcoming period.

References

1. D. M. Rafi et al., Benefits and limitations of automated software testing: Systematic literature review and practitioner survey, In 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, 2012, pp. 36-42.
2. Dustin, E., T. Garrett and B. Gauf. Implementing Automated Software Testing. Boston: Addison-Wesley, 2009, pp. 24-26.
3. Q. Xie and A. M. Memon, Using a Pilot Study to Derive a GUI Model for Automated Testing, In ACM Transactions on Computational Logic, vol. 18, no. 2, Nov 2008.
4. Van der Alast, L. et al. TMap Next: Business Driven Test Management, Nederlands: UTN Publishers, 2008.
5. Koomen, T. et al. TMap Next for result-driven testing, Nederlands: UTN Publishers, 2006.
6. J. J. Haswell, R.J. Young and K. Schramm, System, method, and article of manufacture for test maintenance in an automated scripting framework, US Patent 6 701 514 B1, Mar 2, 2004.
7. C. Ibbs, C. Wong and Y. Kwak, Project Change Management System, In Journal of Management in Engineering, vol. 17, no. 3, Jul 2001.