



Paradigm shift in test automation

The solution to the maintenance problem

Editor: Wolfgang Platz
Authors: Michael Hentze
Wolfgang Platz

Table of contents

1	Introduction	3
2	The maintenance problem.....	4
3	Eliminating maintenance - Dynamic steering.....	6
4	Minimizing unavoidable maintenance	7
5	Summary	8
6	References.....	9

Abstract

Already in the 80s and 90s, it was praised eagerly, even in a premature state, as the ultimate solution: test automation in order to cover the permanently growing test expenses.

Today many companies that tried to implement automated tests themselves are disillusioned. In many cases, the high expectations could not be met: the maintenance efforts of automated tests are often underestimated initially.

This document introduces strategies to eliminate unnecessary maintenance efforts and to minimize unavoidable maintenance expenses. These strategies become possible through a paradigm shift in test automation. This paradigm shift is powered by the TOSCA Testsuite™ of TRICENTIS®.

TRICENTIS® unites the concepts of its dynamic steering concept in TOSCA Testsuite™ under the umbrella of **Business Dynamic Steering**.

Legal notice

TRICENTIS® Technology & Consulting GmbH
Leonard-Bernstein-Straße 10
1220 Vienna
Austria

Tel.: +43 (1) 263 24 09
Fax: +43 (1) 263 24 09-15
E-Mail: office@tricentis.com

© by TRICENTIS® Technology & Consulting GmbH

1 Introduction

The functional capability of software that is constantly developed must be tested again and again. Short product cycles on the market and the increasing complexity of software systems resulted in the continuous growth in demand for testing in the last decades. New legal regulations (Sarbanes-Oxley Act of 2002 (SOX), EU Directive 8, etc.) require compulsory and comprehensive tests before software is rolled out or newly developed and therefore further increase the pressure on the testing departments.

Already in the 80s and 90s, it was praised eagerly, even in a premature state, as the ultimate solution: test automation in order to cover the permanently growing test expenses.

Expectations for test automation **[DHAY01]**:

- Reduced running time
Many computers execute tests simultaneously and with maximum speed twenty four hours per day. Thus the running time of tests is reduced significantly.
- Higher actual test coverage and better quality
Higher test performance per time unit allows to execute large sets of test cases even in short intervals. Comprehensive and detailed error reports in the medium and long term result in improved software quality.
- Significant cost reduction in software testing
Previously manual software tests are now executed by computers. The result is a considerable reduction in costs.
- Complete traceability of the tests
Automated tests must be defined precisely and are characterized by the consistent quality of execution and documentation of the tests. Thus formal software test requirements are met.
- Simple creation of automated test cases
The recording functionality of capture/replay tools allows to conveniently and easily record automated test cases as scripts.

Today many companies that tried to implement automated software tests are disillusioned: in many cases the high expectations could not be met:

- Low coverage in automated testing
Generally the companies only reach a test coverage of a maximum of 30% for complex systems **[TRIC01]**.
- Little or no cost reduction
The expected break-even point of the amortization of investments in automated tests could not be achieved, not even closely.
- Development of complex test frameworks
To keep the scripts executable, expensive customer-specific frameworks must be developed (develop/replay). In addition to the actual software development, technical projects are required to create and carry out automated tests.

All unmet expectations were caused by the same reason that accompanied automated software tests from the outset: as the application itself, the test cases require maintenance. In most cases, the maintenance efforts are significantly underestimated in the initial phase.

In 1997, Cem Kaner [CKAN01] pointed out that test automation by capture/replay does not work and requires to set up appropriate frameworks in order to minimize the maintenance problem. In the following years, Mark Fewster [MFEW01] and many others [EHEN02], [LDIM01], ... confirmed and further specified Cem Kaner's approach. All of them have one thing in common: the recommendation to set up technical frameworks.

This document introduces strategies to eliminate unnecessary maintenance efforts and to minimize unavoidable maintenance expenses. These strategies become possible through a paradigm shift in test automation. This paradigm shift is powered by the TOSCA Testsuite™ of TRICENTIS®.

2 The maintenance problem

The main part of maintenance efforts in automated testing results from the fact that technical source code (technical scripts) is used to document the automated test sequence.

Traditional approaches in test automation (capture/replay, develop/replay) produce and use test scripts in the form of source code. The individual lines in the code represent inputs or verifications. The lines in the code display represent the business task at the time of recording, which means that they are a technical snapshot of the business task.

The maintenance problem is illustrated in the following example:

In an SAP application the last edited billing document must be selected:

Pos	Role	Name 1	Place	BillingDoc
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035560
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035561
<input checked="" type="checkbox"/>	WE	IDES AG	Frankfurt	90035562
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035563
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035564
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035565
<input type="checkbox"/>	WE	IDES AG	Frankfurt	90035566

Illustration 1: Business task, example

To complete this task, a manual tester must take the following steps:

- Find the table of billing documents on the screen
- Assign the relevant billing document number (in this example 90035562)
- Localize this billing document in the table
- Select the row with a click on the CheckBox

This is the script to this business task¹:

```
CheckBox Click, "/usr/cntlCONTAINER/shellcont/shell[2]/chbx[1,3]"
```

1. The syntax of the script was modified and represents a hybrid of syntax types of the traditional automation tools.

Note to the script:

- This instruction cannot be read by the business unit
The business information is not visible. The meaning of the script is: "Click on the Checkbox in the third row and first column of the second table that is displayed on the screen".
- This instruction is unstable from the business point of view.
The business information and its technical specification are only equivalent at the time of recording. At a later time, the click on the checkbox in the third row will possibly execute a completely different business task (as the last edited billing document will have a different number and/or will be located in a different row).

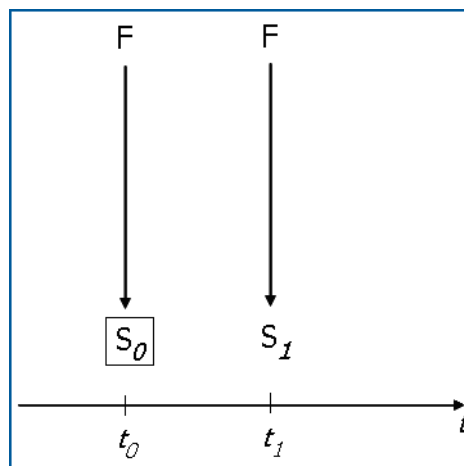


Illustration 2: Chronological sequence of the business task and the technical script

The following applies:

At the time t_0 a business task F produces a technical script S_0 . This script documents the test sequence. The same business task creates the script S_1 at t_1 . If S_0 is executed again at t_1 , then F is not necessarily executed.

To ensure that F is also executed at t_1 , the traditional approaches require a shift from S_0 to S_1 (maintenance problem of type I in automated software testing).

This maintenance can be completely eliminated, if the steering is re-calculated at any time of the test execution (see chapter 3 "Eliminating maintenance - Dynamic steering").

Modifications of the business task are by far more infrequent, but they still cause maintenance requirement in the automated test case (maintenance problem of type II in automated software testing).

This maintenance requirement must be minimized (see chapter 4 "Minimizing unavoidable maintenance").

3 Eliminating maintenance - Dynamic steering

The maintenance of type I can be completely eliminated. To achieve this goal, the automated tests are no longer documented by static, technical source code.

For all instructions, the technical steering must be determined dynamically based on the logical description at the time of test execution (paradigm shift in test automation).

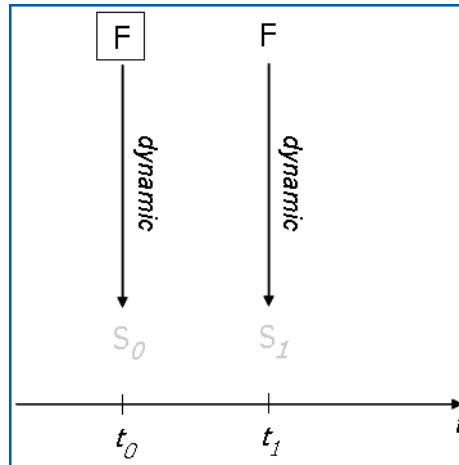


Illustration 3: Dynamic steering

The test sequence is documented logically from a business point of view by **F**. The steering is determined dynamically at the time of execution and does not persist.

A complete dynamization requires concepts for the following tasks:

- Dynamic calculation of steering instructions from business-based descriptions for all commands (see above)

The dynamic steering must work without exceptions for all systems of the company that should be tested, no matter if they are used via GUIs or non-visual interfaces.

- Support of dynamic date and time specifications

Date and time are never specified absolutely in test case specifications, but always relatively. (Examples: the first of the month, the current date, the last of the quarter). To ensure stable test executions from the business point of view, these specifications must be processed at runtime.

- Support of dynamic, logical variables

Some test values are generated during runtime (reference numbers, etc.) or are modified continuously (product definitions, etc.). These must be included in the automated test as variables with meaningful logical names and must be interpreted at runtime.

TRICENTIS® unites the concepts of its dynamic steering concept in TOSCA Testsuite™ under the umbrella of **Business Dynamic Steering**.

TOSCA Testsuite™ by TRICENTIS® puts the requested paradigm shift into practice thus completely solves the maintenance problem of type I.

The task shown in the example (see Illustration 1: "Business task, example") is specified logically, from a business point of view in TOSCA as follows:

Billing Doc	
Aktion	Select
Spalte	
Wert	
Zeile	Billing Doc={B[LastBillingDocNo]}

Illustration 4: Business task, specification in the TOSCA Testsuite™

Illustration 4: "Business task, specification in the TOSCA Testsuite™": LastBillingDocNo is a variable that has the value 90035562 at the time of test case creation, but it can have any values at a later time.

Name	Value	ActionMode
Vehicle-Premiumverification		
Select Vehicle type		
Vehicle type selection		
Type of vehicle	automobile	Input
Goto vehicle data	X	Input
Enter Vehicle data		
Vehicle data		
Brand	BMW 335i	Input
KW	225	Input
Year of manufacture	{Year-7Y}	Input
Seats	5	Input
Type of fuel	gas	Input
Original price	57925	Input
Accept	X	Input

Illustration 5: Business-based automated test case in the TOSCA Testsuite™, extract

Illustration 5: "Business-based automated test case in the TOSCA Testsuite™, extract": {Year-7Y}² represents a dynamic date: due to the business specification, the year of manufacture must be 7 years in the past.

Illustration 4: "Business task, specification in the TOSCA Testsuite™" and Illustration 5: "Business-based automated test case in the TOSCA Testsuite™, extract" show that **automated tests in the TOSCA Testsuite™ can be created and executed without technical know-how**. As a "secondary effect" of the dynamic steering, the test becomes what it should actually be: a business-based task.

4 Minimizing unavoidable maintenance

The maintenance of type II cannot be avoided and thus it must be minimized. The frameworks that must be established when traditional test tools are used, are primarily aimed at the reduction of maintenance of type II.

To minimize the maintenance requirements of type II, the technical and business information must be structured without redundancy. Thus this information is merged so that it can be maintained centrally.

2. TOSCA Testsuite™ is available in German and English.

This redundancy can be eliminated by using modular principles. TOSCA Testsuite™ provides modular principles on three levels:

- ObjectMaps for technical definitions without redundancy³
Technical definitions for screen elements or interfaces are stored centrally
- TestStepLibrary for reusable dialog modules
Dialog sequences that are used multiple times are linked to one original. Modifications are synchronized in all references and only have to be modified in one place.
- Templates to test large data volume
Test sequences are defined centrally and can be linked with a large volume of test data combinations.

5 Summary

The first steps in the history of automated software tests were made 30 years ago. While fundamental innovations were made in the software development industry, the maintenance problems prevented the great breakthrough of test automation - until now.

With its dynamic steering concept, the TOSCA Testsuite™ marks the paradigm shift in automated software test. The dynamic steering is a business-based description of automated test cases without scripts and is globally unique. TRICENTIS® unites the concepts of its dynamic steering concept in TOSCA Testsuite™ under the umbrella of **Business Dynamic Steering**.

This paradigm shift allows to completely eliminate maintenance efforts caused by scripts (type I; this represents the most substantial part of the overall effort, if traditional test tools are used). The remaining, unavoidable business-related maintenance effort (type II) is minimized by the unique modular principles.

The substantial reduction of maintenance efforts allows to multiply the coverage of automated tests: while traditional test tools typically provide a test coverage of 30%, companies using TOSCA can reach about 80%.

Automated tests in the TOSCA Testsuite™ can be created and executed without any technical skills. As a "secondary effect" of the dynamic steering, the test becomes what it should actually be: a business-based task.

As the first solution on the market, TOSCA eliminates the necessity to develop specific frameworks. This leads to the significant reduction of total cost of ownership in automated software testing [EHEN01].

3. A rudimentary form of the concept already exists in traditional test tools.

6 References

[DHAY01] Dawn Haynes, 2001, Automated Testing: A Silver Bullet?

[TRICO1] TRICENTIS[®] Technology & Consulting GmbH, 1997 - 2008, statistical data from customer projects

[CKAN01] Cem Kaner, 1997, Improving the Maintainability of Automated Test Suites

[MFEW01] Mark Fewster & Grove Consultants, 2001, Common Mistakes in Test Automation

[EHEN01] Elisabeth Hendrickson, 2000, Build It or Buy It

[EHEN02] Elisabeth Hendrickson, 2001, Bang for the Buck Test Automation

[LDIM01] Len DiMaggio, 2001, Bringing Your Test Data to Life