# Paradigm Shift in Test Automation

## The Solution to the Maintenance Problem

Author: TRICENTIS Founder and CEO
Wolfgang Platz

# 1 Executive Summary

As early as the 80s and 90s, test automation was widely lauded as the **silver bullet** for the ever-growing cost of testing.

Today, many companies have become disillusioned with automated testing. The initial high expectations have not been met, and the maintenance effort for automated testing has often been greatly underestimated.

This document introduces strategies to eliminate unnecessary maintenance effort and to minimize unavoidable maintenance cost. These strategies are made possible by a paradigm shift in test automation powered by TOSCA Testsuite of TRICENTIS.

TRICENTIS unites the concepts of its dynamic steering approach in TOSCA Testsuite under the umbrella of *Business Dynamic Steering*, which we will explore in this white paper.

## Table of contents

## 2 Introduction

The functional capability of software that is constantly modified and enhanced must be tested repeatedly. Shorter product cycles on the market and the increasing complexity of software systems have resulted in an ever greater demand for testing over the last several years. New regulatory frameworks, for example, the Sarbanes-Oxley Act of 2002 and the 8th Company Law Directive of the EU, mandate comprehensive testing before software is rolled out or updated, and further increase pressure on testing departments.

As early as the 80s and 90s, test automation was widely lauded, although prematurely, as the 'silver bullet' for the ever-growing cost of testing, and this created expectations, in the test community and management, that test automation would **[DHAY01]**:

- Reduce running time

  Multiple computers would execute tests simultaneously, at high speed, twenty four hours a day, significantly reducing the running time of tests.

- Achieve higher actual test coverage and better quality.

  Higher test performance rates would allow the execution of large sets of test cases in a short time. The resulting comprehensive and detailed error reports would yield improved software quality in the medium to long term.

- Significantly reduce software testing costs.

  What was previously done manually would be executed by computers, considerably reducing costs.

- Allow 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 satisfying formal software test requirements.

- Simplify the creation of automated test cases

  The recording functionality of capture/replay tools would allow the convenient and easy recording of automated test cases as scripts.

Today's reality is that many companies are disillusioned with test automation, because these high expectations have often not been met, with the following outcomes:

- Low coverage in automated testing

  Generally, companies only attain test coverage of around 30% for complex systems **[TRIC01]**.

- Little or no cost reduction

  Companies have not even come close to achieving the expected amortization of investments (break-even points) in automated testing.

- Development of complex test frameworks

  The deployment of automated testing has not only brought with it the overhead of developing and maintaining expensive customer-specific test frameworks, but has also resulted in additional projects to create and execute automated tests.

These poor outcomes are due to a problem that has plagued automated software testing right from the beginning: as with the application software itself, the emerging test environment (test cases, test scripts, test execution frameworks etc.) requires maintenance, and the maintenance effort, in most cases, has been significantly underestimated at the outset.

Cem Kaner **[CKAN01]** was already pointing out in 1997 that test automation by capture/replay does not work and requires appropriate frameworks to be set up in order to minimize the maintenance problem. In following years, Mark Fewster **[MFEW01]** and others **[EHEN02], [LDIM01] etc.** verified and further refined Cem Kaner's approach. All of them have one thing in common: they recommend setting up test frameworks.

This document introduces strategies to eliminate unnecessary maintenance effort and to minimize unavoidable maintenance cost. These strategies are made possible by a paradigm shift in test automation powered by the TOSCA Testsuite of TRICENTIS.

The following illustration compares the productive efficiency of TOSCA Testsuite with traditional approaches:

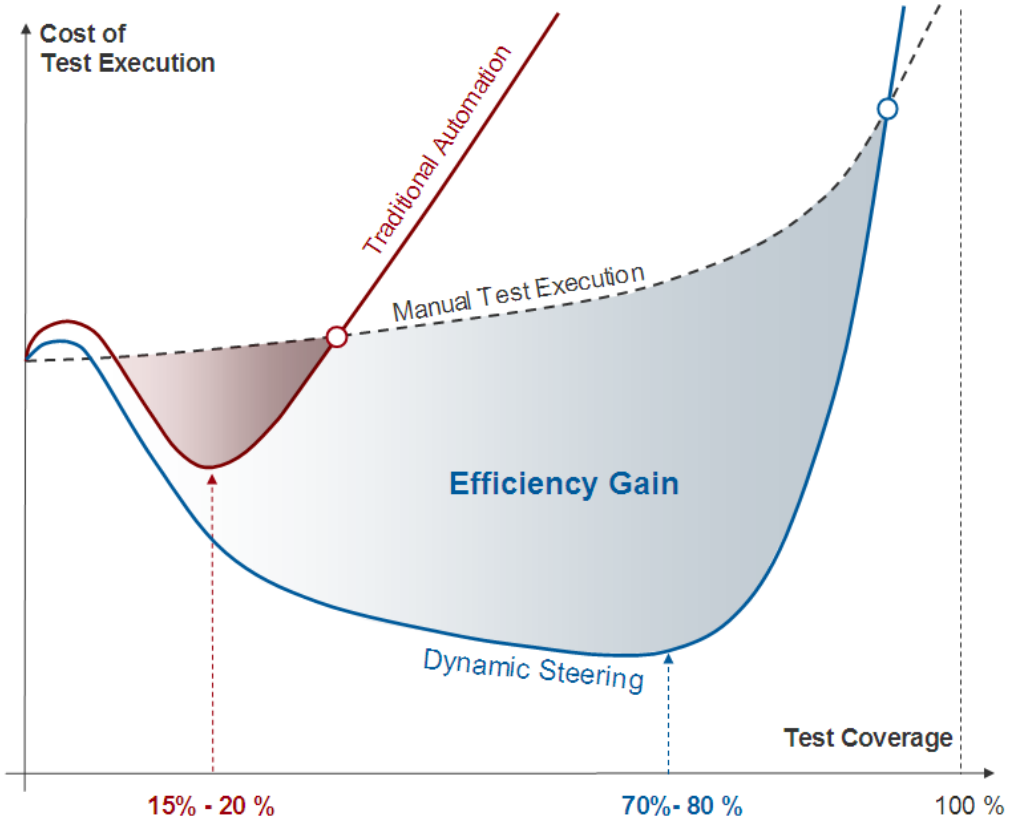


Illustration 1: Test automation with TOSCA compared to traditional approaches

**Notes on Illustration 1**

- Traditional approaches require the setup of a test framework and thus an initial development investment. As a result, the initial costs of creating and running automated tests are significantly higher than the costs of manual test execution.
- An increasing number of test cases (resulting in increased test coverage) leads to some economies of scale with traditional automation - up to a point. Within a certain range of test coverage, savings can be achieved compared to manual tests (indicated by the red area).
- Optimal results can be achieved using traditional approaches in a bandwidth of test coverage ranging from 15% to 20%, but not exceeding about 30%. Beyond this mark, unit costs of automated test cases usually rise quickly, owing to the increasing maintenance effort, and the cost line of manual test execution drops below the traditional automation line.
- With the Dynamic Steering approach, TOSCA Testsuite solves the maintenance problem of test automation, and introduces a paradigm shift in the test industry.

- Initial investments are considerably lower than those of traditional approaches because there is no need to build test frameworks. The average costs of automated tests decrease rapidly with increasing test coverage, and reach a significantly lower level than could ever be achieved with traditional approaches - the economies of scale persist to much higher test coverage levels.
- Using TOSCA Testsuite, our customers report optimal results at test coverage levels of around 70% - 80%, which is made possible by test automation.
- The efficiency gain (indicated by the blue area) is many times that of traditional approaches.

The following sections will give a detailed explanation of the maintenance problem, and the specific solution provided by TOSCA Testsuite.

# 3 The maintenance problem

The major maintenance effort in automated testing results from the use of source code (usually in script form) to document the automated test sequence.

Traditional approaches to 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 represent the business task at the time of recording, and are a programmatic snapshot of the business task.

The maintenance problem is illustrated in the following example:

In an SAP application, the most recently edited billing document must be selected:

| Pos | Role | Name 1 | Place | BillingDoc |
|---|---|---|---|---|
| | WE | IDES AG | Frankfurt | 90035560 |
| | WE | IDES AG | Frankfurt | 90035561 |
| | WE | IDES AG | Frankfurt | 90035562 |
| | WE | IDES AG | Frankfurt | 90035563 |
| | WE | IDES AG | Frankfurt | 90035564 |
| | WE | IDES AG | Frankfurt | 90035565 |
| | WE | IDES AG | Frankfurt | 90035566 |

Illustration 2: Example of a business task

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

- Find the table of billing documents on the screen
- Assign the relevant billing document number (in this example, 90035562)
- Locate this billing document in the table
- Select the row by clicking the check box

This is the script for this business task[1]:

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

**Notes on the script:**

- This instruction cannot be read by the business unit

[1] The syntax of the script has been altered and represents a hybrid of syntax types of traditional automation tools.

The business information is lost. The meaning of the script is: **Click on the check box in the third row and first column of the second table that is displayed on the screen**.

- This instruction is only temporary from the business point of view.

  The business information and its technical specification only match at the time of recording. At a later time, clicking on the check box in the third row might execute a

  completely different business task (as the most recently edited billing document will have a different number and/or will be located in a different row).

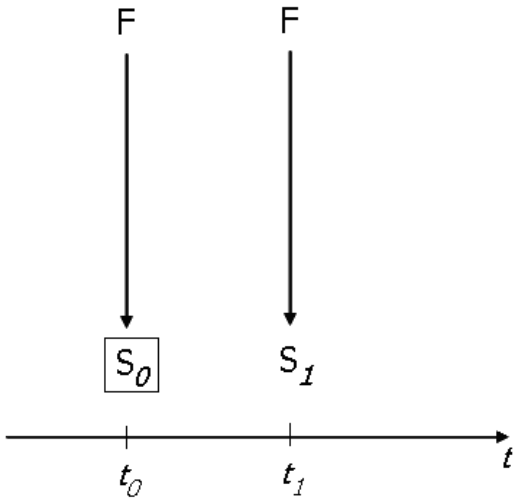


Illustration 3: Chronological sequence of the business task and the technical script

**Notes on Illustration 3:**

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 $S_0$ were executed again at $t_1$, then **F** would not necessarily be carried out.

To ensure that **F** is also executed at $t_1$, traditional approaches require $S_0$ to be converted into $S_1$ leading to a **type I** maintenance problem in automated software testing.

Maintenance of this type can be completely eliminated if the script parameters are recalculated every time the test is executed (Eliminating maintenance - Dynamic steering).

Modifications of the business task are far more infrequent, but are still a source of change in automated test cases: a problem we refer to as a type II maintenance problem in automated software testing.

This maintenance requirement, though unavoidable, should be minimized (Minimizing unavoidable maintenance)**.**

# 4 Eliminating maintenance - Dynamic steering

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

For all test instructions, the path through the test, and associated parameters (that is, how we steer the test), must be determined dynamically based on the logical description at the time of test execution - and this is the paradigm shift in test automation that we call *dynamic steering*.

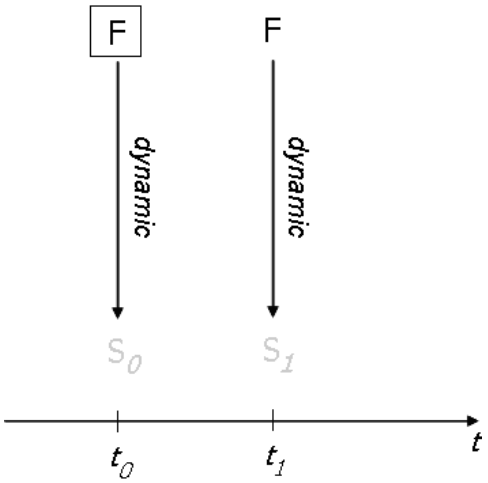


Illustration 4: Dynamic steering

The test sequence is documented logically from a business point of view as part of **F**. How the path is steered is determined dynamically at the time of execution and does not persist.

There are some prerequisites to complete dynamization:

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

  Dynamic steering must be made to work, without exception, for all systems of the company that are to be tested, regardless of whether 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, for example, the first of the month, the current date, or the last day of the quarter. To ensure stable test execution from the business point of view, these specifications must be processed and filled with actual values at runtime.

- Support of dynamic, logical variables

  Some test values are generated at 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 processed and filled at runtime.

TRICENTIS unites the concepts of its dynamic steering approach in TOSCA Testsuite under the umbrella term of *Business Dynamic Steering*, thus emphasizing that testing proceeds from a business perspective, and that the derivation of the path through a test - the instructions and parameters, or how we steer the test - is *dynamic*, that is, determined at the time of test execution.

TOSCA Testsuite by TRICENTIS puts the required paradigm shift into practice, and thus completely solves the type 1 maintenance problem, as shown in the following illustrations.

The task shown in the example (Illustration 2: Example of a business task) is specified logically, from a business point of view in TOSCA, as follows:

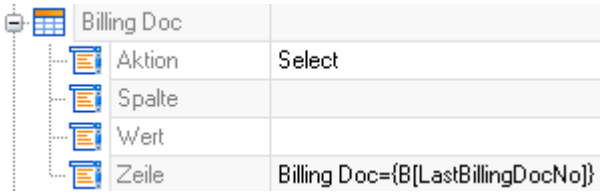


Illustration 5: Specification of a business task TOSCA Testsuite

In Illustration 5 the **LastBillingDocNo** is a variable that had the value **90035562** at the time of test case creation, but which can assume other values later on.

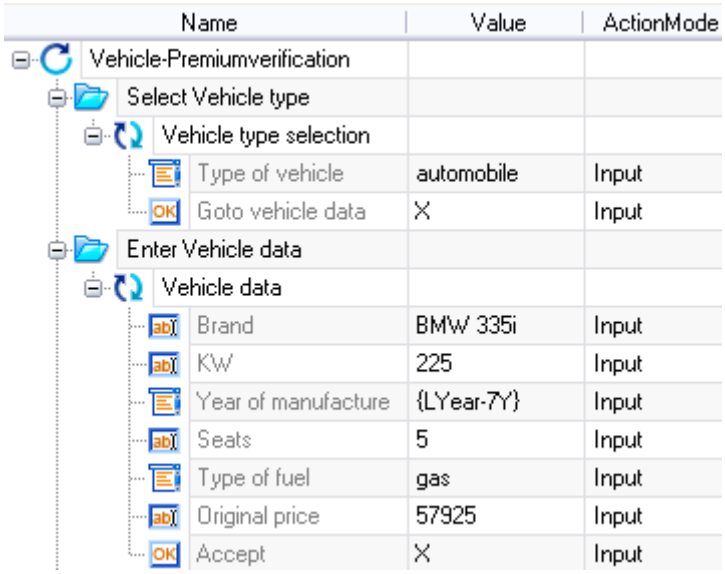


Illustration 6: Extract from a business-based automated test case in the TOSCA Testsuite

In Illustration 6 we can see that **{LYear-7Y}**[2] represents a dynamic date. According to the business specification, the year of manufacture must be **7** years ago.

Illustration 5 and Illustration 6 show that **automated tests in TOSCA Testsuite can be created and executed without technical know-how**. As a side benefit of dynamic steering, the test becomes what it should always have been: a business-based task.

# 5 Minimizing unavoidable maintenance

Maintenance of **type II** cannot be avoided, but it can (and obviously should) be minimized. The frameworks that must be established when using traditional test tools are primarily aimed at the reduction of **type II** maintenance.

To minimize type II maintenance requirements, technical and business specifications should be structured without redundancy - that is, we want to avoid having multiple copies of the same specification that differ only in the level of abstraction. Ideally, this information is combined in such a way that it can be maintained centrally.

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

- ObjectMaps for non-redundant technical definitions[3]

  Technical definitions for screen elements or interfaces are stored centrally

- TestStepLibraries for reusable dialog modules

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

[3] A rudimentary form of this concept already exists in traditional test tools.

Dialog sequences 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 a large data volume

  Test sequences are defined centrally and can be linked with a large volume of test data combinations.

# 6 Summary

The first steps in the history of automated software testing were taken 30 years ago. While fundamental innovations have been made in the software development industry, maintenance problems have stymied the great early promise of test automation - until now.

With its dynamic steering concept, the TOSCA Testsuite marks a paradigm shift in automated software testing. 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 approach in TOSCA Testsuite under the umbrella term of **Business Dynamic Steering**.

This paradigm shift allows for the complete elimination of maintenance efforts caused by scripts - what we have called type I maintenance, representing the most substantial part of the overall effort if traditional test tools are used. The remaining, unavoidable business-related maintenance effort, what we have called type II maintenance, is minimized by the unique modular principles of TOSCA Testsuite.

This substantial reduction of maintenance effort enables a multiplication of the coverage of automated tests: while traditional test tools typically provide test coverage of around 30%, companies using TOSCA can routinely reach up to 80%.

Automated tests in TOSCA Testsuite can be created and executed without any technical skills. As a side benefit of dynamic steering, the test becomes what it should always have been: a business-based task.

The only one of its kind on the market, TOSCA eliminates the necessity of developing specific frameworks. This leads to a significant reduction of the **total cost of ownership** in automated software testing **[EHEN01]**.

# 7 Appendix

It is in connection with this paradigm shift that TRICENTIS refers to its test cases as “cognitive test cases”. In using the term “cognitive” to describe test cases, we wish to emphasize certain aspects of test case specification in TOSCA, which are based on the theory and findings of cognitive science. In cognitive science, cognition (human thought processes) is understood to consist of internal mental states (representations), which are processed like data structures in a digital computer. Human cognition, like a digital computer, manipulates the representations by means of rules or algorithms, which contain the instructions for the procedure within themselves. Thus, one could say that cognition processes representations like a cook following a recipe. In this way, human thought processes could be said to be self-contained, that is, they do not require any external or secondary source to process the representations.

As we have seen above, the dynamic steering approach of TOSCA not only means that testers do not require test frameworks to execute test cases, but they do not even need any test scripts in the form of source code. Owing to the modular structure of TOSCA, testers build test cases by dragging and dropping modules, which contain all the technical information of the application, regardless of the interface (GUI and non-GUI), and then entering validation values and actions. Instead of manipulating test scripts and creating test execution frameworks, TOSCA testers use modules to build sequences of test case steps with descriptive names, thus making TOSCA self-documenting. Since testers do not have to read and understand code to create and structure test cases, but simply drag and drop modules, the cognitive fluency of the system is increased, which makes it easier and quicker for subject matter experts (SMEs) to learn how to create and structure test cases in TOSCA. Thus, test case specification in TOSCA is no longer a technical, but a business-based task. In addition, since manual and automated test cases are created in the same way and can be integrated into the same test sequence, the automation of test cases is easier and quicker and thus the degree of automation is increased exponentially, particularly in the case of regression tests.

Thus, when TRICENTIS speaks of “cognitive” test cases, it means that the specification of test cases is a business-based task founded upon cognitive principles and that the test cases are self-contained and self-documenting, which leads to greater cognitive fluency as well as higher levels of automation.

## 8 References and sources

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

**[TRIC01]** 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

***