Automatic maintenance of test scripts

Maria Ana Casal Cunha

November 21, 2012

Abstract

Currently, it is generally accepted by the software industry the importance of test automation in the process of quality assurance of their products. However, a major problem that this technique faces is the effort required for the maintenance of test scripts.

This problem was experienced by the testers of the development project of the NextWay web platform. In this project, due to lack of stability concerning the application requirements, testers spent a large amount of their time upgrading the old test cases for the new version of the application. Thus, it became necessary to implement a tool that could support a tester in the maintenance of his test scripts to reduce the time spent in that process.

The solution developed is based on the identification of differences between two versions of the GUI of the application in order to reflect this information in the test scripts, restoring the correspondence between them. It is intended that the effort required from the tester in the test maintenance process is the minimum possible. With the test maintenance tool developed, the tester is now capable to repair his test scripts in a very efficient way. The tool is able to automatically maintain the tests in most cases, although there are some cases that require manual intervention to repair a test.

1 Introduction

Software Testing is an important area of software engineering and consists in a process of analysing the system under test in order to reduce the defects of its implementation. An important goal of software testing is to detect faults in the system under evaluation as soon as possible so that defects are corrected with the least cost. It is important that software testing is a continuous activity along with the development process [1].

Software Testing can be performed at various levels, from simple components, integration of components to the level of the graphical interfaces where the system can be tested as a whole. Regardless the level on which software testing is performed, its manual execution is always a repetitive and boring process for testers. Moreover, test execution has a cost in money and time. Also, when the application under test is complex and its development process is agile (where tests have to be executed at each new iteration) these problems assume bigger proportions. Thus, in this context the need for automation of tests is even higher.

Concerning test automation, there is an added cost in preparing a script for automatic testing, but the extra effort in its development is recovered when the script is repeatedly executed over several cycles of testing[9, 8].

In this sense, there has been a large investment on Software Quality and innovations have been made to improve the techniques for the development and generation of test scripts using automation tools. However a major problem arises with the automation of test scripts, which is their maintenance. Since they are intrinsically related to the application under test, the minimal change in the application can lead to the invalidation of the test scripts created so far[7].

When the application under test is modified, one of three situations can happen to a test script:

- The script does not need maintenance (because it's not affected by the changes in the application);
- The script becomes obsolete (because it doesn't make sense any more);
- The script becomes broken (because it still makes sense but needs maintenance).

When the application under test is still unstable and it's modified frequently, the costs of maintaining the automated test scripts can grow until they become unbearable, leading project managers to abandon this approach, returning to manual testing only[10].

In this paper it is presented a new approach to the process of maintaining test scripts so they remain valid for testing new versions of the application under test, requiring the least effort possible by the tester.

1.1 Context and Problem

The solution presented in this paper was developed within the context of support for quality as-

surance of the Web platform, NextWay.

Aiming to develop a quality product, the NextWay project had from the beginning a Software Quality team, which had the responsibility to develop a solution for automated functional regression tests using AnyTester, an automation testing tool which is Indra's property.

AnyTester is a test automation tool for Web applications that models test scripts through an intuitive graphical interface that requires only the business functional knowledge from the user. It runs tests at the GUI level of the application under test in order to simulate the actions and validations performed by a tester. It uses Selenium[1] as the engine for automatic execution of its test scripts.

In the NextWay project, it was intended to have a test suite that would run continuously in order to detect, as soon as possible, any regression that could happen in the platform under test. At each new functionality implemented, the test team should perform the following tasks:

- Create the test scripts that exercise that functionality;
- Run the test suite in order to verify if any regression occurred;
- Repair any test script, previously developed, that could be affected by changes in the GUI of the platform.

In the first months of the project, the platform suffered many changes in its GUI, which led to a constant break in its correspondence with the test scripts created so far. Initially, as the number of tests created was still small and the frequency of changes to the GUI was not high, it was possible to maintain the test scripts manually.

However, with the growth of the platform, the cost of this effort became increasingly unbearable to the point that there were 755 test scripts, all invalid and mismatched with the platform. This problem led the Project Management to abandon the test automation solution, assuming the risks associated with this decision.

Thus, and taking into account the importance of automatic regression testing, it arose the need to develop an extension for AnyTester that would enable the maintenance of test scripts, with the least possible effort from the tester.

1.2 Solution

In order to mitigate the impact that changes on the graphical interface of the platform may have on the test scripts, it was intended to develop a solution to repair the broken test scripts (requiring the minimum effort from the tester).

[1] http://www.seleniumhq.org/

This solution should be based on the identification of changes that occur between two distinct versions of the platform:

- A new page is developed;
- A page is removed from the platform;
- In one page there is a new web element;
- In one page a web element is removed;
- In one page a web element is modified.

Ideally it was intended to implement a solution that would be capable of maintaining all tests affected by changes in the GUI of the platform without requiring any manual intervention by the tester. Situations like the elimination of a page from the platform or the removal of an element on a page without any other being added, should correspond to changes completely transparent for the tester. In the first case, the tests that navigate through the erased page should be automatically removed from the test suite and in the second case, the actions that focus on the element removed should also be removed from the test.

However, there are situations in which the intervention of the tester could be necessary. For example, when the identifier of a given element is changed, the maintenance tool should detect that there is an element that has been removed and another that was been added in that page. In certain situations the tool should be able to infer that the removed element corresponds to the added element in the new version of the software, but in other situations it will have to be the tester to provide this information.

After finishing the entire process of analysing the changes undergone by the platform in terms of its graphical interface, the maintenance tool should start to repair the test scripts as follows:

- Correctly updates the name of the elements changed, replacing the old with the new one.
- Removes the actions on deleted elements;
- Marks the actions that reference removed actions;
- Invalidates all scripts that navigate through pages that have been deleted from the platform and removes them from the test suite.

At the end, a report should be presented to the tester. It is considered that it's important to provide this report so that the tester can check the changes that were made in the test scripts. In addition, through this report the tester is informed of other modifications in the platform, such as, the introduction of new elements on certain pages referenced in the test scripts and the development of

new pages to which it may be necessary to develop new test scripts.

This solution is based on the perception of how the GUI of the platform has evolved between two versions and then use that information to repair the test scripts. Thus the effort of the tester, if necessary, should be focused on a single point and all information collected is subsequently reflected on all the test scripts. The tester has to identify the transformation of one element only once and then all the test scripts that reference that element are automatically repaired.

1.3 Outline

The remainder of this article is organized as follows. Section 2 gives account of the related work on test scripts maintenance. On section 3 it is made a description of the test automation tool AnyTester. On section 4 it is presented in more detail the solution for the maintenance of test scripts. Section 5 presents the results obtained during the evaluation process of the implemented solution and on section 6 it is made the final remarks.

2 Related Work

Regardless of the level at which the regression tests are performed, it is common to automate this process with the aid of tools that allow the construction, analysis and execution of test scripts. However, it is natural that in the course of the software development process, changes made by programmers in the application break its correspondence with the test scripts, making them unable to test new versions of that application.

Some techniques have been developed to mitigate this problem, avoiding that the effort made in the automation of a test for a particular version of an application is wasted at the launch of a new one.

In terms of the maintenance of unit tests, Daniel Brett and his colleagues present ReAssert [3], a solution that suggests repairs for tests that fail due to changes in software. It is based on a combined dynamic and static analysis to the broken test to find repairs that might be accepted by the programmers. These suggestions include updating expected values with actual values obtained from running the broken test, changing assert methods, replacing one assert method by a set of them and others. The programmer can then compare the original test with the suggestions made by the tool and, if he realizes that indeed these are the changes that he wants to do, with a click on a button they are performed.

Mark Grechanik presents REST [6, 5], an approach for maintaining and evolving test scripts for applications with graphical user interfaces,in a black box regression testing scenario where the source code is not available. This solution is based on modelling the GUI in a tree representation, in which objects are represented in a hierarchy representing its layout on the page. Then, based on these models, the tool identifies the differences between the GUIs of two successive versions of the application. Using this comparison, the test scripts are analysed to detect how they were affected by the changes in the application under test. In the end, a report is generated with a series of warnings that allow testers to know which corrections they have to manually perform on their test scripts.

Choudhary and his colleagues present WATER [2], a black box solution that makes suggestions of repairs on test scripts in applications with GUIs. It identifies three causes for the failure of tests between two versions of the application:

- Structural changes: where changes occur at the level of the tree structure that corresponds to the HTML code of a page;
- Content changes: where the content of a given web element is changed;
- Blind changes: changes on the business login of the application, which are not taken into account by WATER.

Once detected the failure point of a particular test script and collected the necessary data, WATER applies the process of determining the list of repairs to be suggested to the tester. These are suggestions for correcting web elements locators, in the case of structural changes, for modifying expected values, in case of changes in content, among others.

Comparing WATER with REST, there are some differences in how the problem is addressed by these two solutions. While REST makes a static analysis of the extracted models of GUIs of two versions of the application, WATER proposes a dynamic collection of data by running the test against both versions of the software to be able to detect the failure point of the test and the best way to fix it. In the end, REST generates a list of warnings of the tests that may be affected by the changes undergone by the GUI, which implies further intervention by the tester to make the necessary corrections. On the other hand, WATER provides a list of suggestions for correction of tests that can be accepted by the tester or not.

In a position paper [4], published in 2011, Brett Daniel extends his research work proposing a solution that combines the automation of GUI refactorings with the process of repairing test scripts. According to the authors, it is possible to automate GUI refactorings, that by their definition correspond to changes only on the visual aspects of the

GUI, in order to reduce the effort of programmers and simultaneously collect the necessary information for updating the test scripts.

3 Context

3.1 AnyTester

AnyTester is a test automation tool, developed by Indra. This application was born from the need for a regression testing solution capable of continuously monitoring the development process of Indra projects.

Using Selenium[1] as the engine for the execution of its test suites, AnyTester provides an intuitive graphical interface that gives the tester, even if he doesn't have a technical profile, the possibility of creating, editing and executing test scripts.

AnyTester allows the tester to define test scripts by specifying a navigation flow through the pages of the application, where it is possible to click and fill elements of a page with values indicated by the tester in the script. It is also possible to check at any point of the flow if the current page is the expected one or if a given element has the desired value.

A test script is created by defining a flow of AnyTester actions, towards rebuilding step by step the actions of a user. The main set of actions provided by AnyTester are:

- Include another test

  With this action, another test is included in that point of the flow.

- Change to Application

  The tester can specify in which application of the platform the next actions will be performed.

- Go to Web Page

  The tester can specify to which page he wants the test to go in that point of the flow.

- Assert Current Page

  The tester can check which is current page at that point of the test flow.

- Set Web Element

  An element is filled with a certain value, both specified in that action.

- Click Web Element

  The element specified in that action is clicked.

- Assert Web Element

  The tester can assert the properties of the element specified in that action.

AnyTester can be executed from the GUI or through a command line. The first provides the tester with an IDE for creating, editing and execution of tests and test suites. The second allows the execution of test suites in automatic processes.

3.2 Test Language Generation

When a tester is creating a test script, AnyTester provides the list of all pages of each application of the platform and all the elements in each of these pages. Thus the tester can specify actions on those elements, such as clicking, fill and check. It is intended that this list is always consistent with the state of the platform so that, whenever new features are developed, it is possible to create new test scripts for those features.

For AnyTester to be able to give this kind of support to the tester, it was implemented (in the scope of the present work) a generator of all the information on the constitution of each application of the platform. This generated information is called Test Language.

The generator of Test Language is a transformer developed in Java[2] technology. It takes, as input, information on the pages of an application. This information is specified by the programmers as part of the development process.

As a result, it returns an XML file for each application of the platform. That file contains the list of all pages and, for each page, a list of all its elements and their properties (for example, the unique identifier).

At each new version of the platform it is necessary to update the information that AnyTester has about it. Thus it's necessary to get the latest version of the source code and then execute the automatic generation of the Test Language of each application. Thereafter, that information is integrated into AnyTester.

4 Solution

Aiming to ensure the quality of the NextWay platform, it was introduced a process of automatic regression testing cycles that, at every new version of the platform, was able to verify the proper functioning of it. However, for this process to have the ability to survive the changes suffered by the platform, it was implemented a solution to ensure the maintenance of the test scripts created so far (with the least possible effort from the tester). Changes

[1] http://www.seleniumhq.org/

[2] http://www.java.com/

in the business logic of the platform under test are not covered by the implemented solution.

Previously, if a particular page had suffered changes in its graphical interface, the tester had to (manually) identify those modifications, identify which test scripts were affected by those modifications and then correct them accordingly. All these tasks were performed without any support from the test automation tool.

Thus, the implemented solution for the maintenance of test scripts is based on the detection of the changes undergone by the GUI in each new version of the platform:

- Which are the new pages;
- Which are the pages deleted from an application;
- Which are the new elements in a page;
- Which are the elements removed from a page;
- Which are the modified elements in a page;

After the identification of the transformations undergone by the platform, all the affected test scripts are automatically analysed and some of them will be automatically repaired while others will require manual intervention.. At the end of this process, our solution will produce a set of test scripts consistent with the new version of the platform.

In the end, a report is generated with all the modifications undergone by the test scripts and informations about other modifications in the platform, such as the introduction of new elements in certain pages and the development of new pages to which it may be necessary to develop new test scripts.

4.1 Examples of modifications in the applications under test

During the NextWay project, whenever the correspondence between a test script and the platform was broken two types of solutions were possible:

- abandonment of the test script, now invalid, and the development of a new one from scratch;
- manual repair of the test script.

Any of those solutions required a great effort on the part of the tester.

From the set of identified modifications to the GUI of the platform, these are the ones with major impact on the test scripts:

- *A web element is modified:*

  - The unique identifier of a web element is modified. A web elements is referenced in a test script through its unique identifier. In case that identifier is modified all test scripts that reference that element stop working. Without our solution, if a tester wanted to repair those test scripts, he would have to find all the references to the element's old identifier and replace it with the new one.

  - The type of a web element is modified. It may also occur situations where the type of an element is modified. In some cases, the transformations can be completely transparent to the tester, like changing from Button to Link, since the actions are equal on both types of web elements. However if the transformation is such that the way to interact with the element has to change, like from Text Box to Select. In this case, the tester would have to manually repair those tests.

  - The text of a web element is modified. There are situations where the text of a Label or a Button change in the new version of the software. In case there is a test scripts that verifies that text, it will fail in its execution. In this situations the tester would have to analyse if that modification is right or if it's an error. In the first case he would have to repair the script, replacing the old text with the new one. In the second case he would have to report the situation to the development's team.

- *A web element is removed from a page.* If a web element is removed from a page, all the test scripts that reference that element stop working. In this case, the tester would have to identify all those test scripts and manually delete all the actions that reference the removed element.

- *A new element is introduced in a certain page.* It is also common that new elements are introduced in certain pages of an application. This case may not even have an impact on existing test scripts. Tests can continue to run and to be successful if no business rule has been changed in a way that would invalidate them, for example, the introduction of a required field in a form. However if the new element causes a test to fail, the tester would have to open it through AnyTester and introduce the necessary actions.

- *A certain page is removed from the platform.* When a page is removed from the platform, all test scripts that navigate through that page become immediately invalid. In such cases, the tester would have to identify which test scripts were affected by that change and manually remove them from the test suite.

4.2 Architecture

In the Figure 1 is presented the architecture of the solution implemented for the maintenance of test scripts. Consider that at a certain stage of the development process of the NextWay project, the software is at version N, AnyTester has the Test Language corresponding to that version and test scripts are also consistent with the software. Later, in version N+1, a developer makes changes to the source code and, as a consequence, the correspondence with the test scripts is broken.

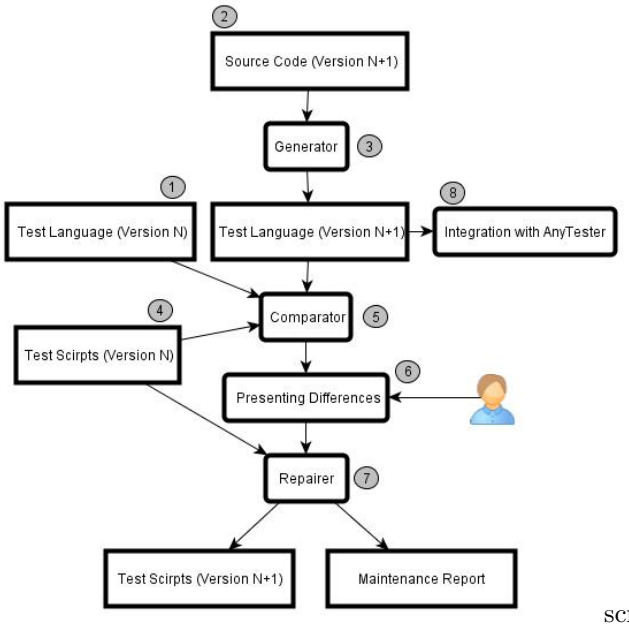

Figure 1: Architecture of Maintenance of Test Scripts Solution.

It is intended that when such a situation is detected the tester starts with the process of maintenance of the tests. To do this, he should run the program responsible for sequentially performing the following tasks:

- Storage the Test Language of version N of the platform (Figure 1 - 1);
- Updating the source code to its latest version, N+1, using the version control system (Figure 1 - 2);
- Generation of Test Language for version N+1 of the platform (Figure 1 - 3);
- Obtaining the list of pages accessed by the test scripts (Figure 1 - 4);
- Comparison of Test Languages, version N with version N+1 (Figure 1 - 5).

In the end the system presents the tester the comparison result of the Test Languages of the two versions of the platform through a graphical interface (Figure 1 - 6). The tester can view, per page, which elements are new and which ones have been eliminated and relate them. In establishing this relationship, the tester informs maintenance tool that a particular element, detected as new in version N+1, corresponds to an other one, detected as removed from version N.

At the end of his analysis, the tester can start the process of repair the test scripts using the option available in the same GUI (Figure reffig:Arq - 7). The repair of the test scripts is done automatically as follows:

- Updating the identifier of changed elements, replacing the old with the new;

  If the element type also have been changed, it may be necessary to also correct the way the interaction with it is specified (eg, in the case of a Text Box becomes a Select Box, and vice versa).

- Removal of actions that reference eliminated elements;
- Marking actions that reference eliminated actions;
- Invalidation of all test scripts that navigate to pages that have been removed from the platform and removing them from the test suite.

Before changes are actually reflected in test scripts, the system shows the tester all the repairs that will be made in each test script, offering him the possibility to confirm or cancel them. By confirming, the test scripts are repaired and a report is presented to the tester with the following information:

- Description of all changes made on each test script (actions removed, changed or marked because they made references to other actions that were removed);
- List of all test scripts where it was identified the existence of new elements on the pages they access and the list of those elements.
- List of all test scripts that access pages that have been deleted from the platform and, therefore, were automatically removed from the test suite.
- List of all the new pages in the platform.

In the end, the tester should start the process of integrating the Test Language of version N+1 into AnyTester (Figure 1 - 8) and then the updated testes are executed against the new version of the platform.

4.2.1 Comparator

The Comparator (Figure 1 - 5) is the component responsible for making the comparison between the Test Languages of two versions of the platform, in order to detect differences in their GUI's.

This component has receives as input:

- Test Language of version N of the platform;
- Test Language of version N+1 of the platform;
- The list of pages accessed by the test scripts.

The tester can decide if he wants all the pages of the platform to be analysed or only the ones referenced by the test scripts. On the one hand, information on the state of an application during the project may be important, for example, for traceability issues, on the other hand it may be in the interest of the tester that only the pages accessed by the test scripts are analysed, since only changes on them may result in direct impact on tests.

In the end the Comparator generates, for each application of the platform, an XML file with information about the differences detected:

- List of pages that exist in both version and for each of these pages:
  - List of new elements;
  - List of deleted elements;
  - List of elements that exist in both versions;
- List of deleted pages;
- List of new pages;

4.2.2 Assisted Approach

After the changes between the two versions of the platform are identified by the Comparator, a graphical interface is presented to the tester where he can view in detail, per page, which elements were removed and which were added.

At this point of the process, the situations in which the identifier of an element is modified has not yet been identified. In such a situation, the analysis of the Comparator results in the detection of one removed element and the detection of a new element, but in reality they correspond to the same one. To repair the test scripts correctly, given such transformations, the tool allows the tester to match removed and new elements, from the same page.

The tester can make the correspondence between elements of the same type or of different types as follows: Button with Link (and vice versa), Check Box with Radio Button (and vice versa) or Text Box with Select Box (and vice versa).

Theoretically, an element may be replaced by an other of any type. However, depending on the transformation, the process of repairing the tests may have to be different. The solution developed supports only these situations because they were considered the most common. For occurring with less frequency, or not occurring at all, no effort has been invested in supporting another kind of transformations.

To facilitate the task of the tester, the tool also provides suggestions about the relationship between the elements added and removed. If an element removed and a new element have the same associated Label it is likely that the second corresponds to the first new version of the platform.

After analysing the evolution of the platform under test between versions N and N+1, the tester should trigger the process of repairing the tests. The Repairer (Figure 1 - 7) is the module responsible for detecting and making the necessary changes in each test script. Before the test scripts are actually modified, it is displayed a preview of the changes scheduled for each of them. If the tester confirms them, the Repairer is responsible for, in fact, performing those modifications on the broken tests.

If a particular test script is identified as obsolete by the tool for navigating through pages removed from the platform, it is automatically removed from the test suite.

In the end a report with the registry of all changes made in each test script and other information about the evolution of the platform is generated and the test suite is executed.

4.2.3 Automatic Approach

After validating the implemented solution it was possible to verify a drastic reduction in the effort required to make the appropriate maintenance of test scripts, making them valid to test a new version of the platform. However, it was felt that the effort of the tester in the maintenance process, to confirm the elimination of actions that reference removed elements and to identify the transformation of certain elements, could be reduced.

Therefore, it was implemented a second approach with the objective of introducing the ability to recover as many test scripts as possible, without any assistance from the tester.

As the assisted approach, at the time the software under test undergoes some evolution that breaks his correspondence with the test scripts, the process of maintaining them should be started. The Test Languages of the two versions of the platform are analysed by the Comparator. When the result of this analysis is presented to the tester, there is an option to trigger the automatic process and recover the maximum number of test scripts without requiring any interaction on the part of the tester.

When choosing this option, the differences between the two versions of the software are analysed in order to infer which elements were, in fact, removed and which ones have changed. Afterwards the test scripts are repaired accordingly.

However, there are still situations where the tool is not able to identify whether a particular item was, in fact, removed or replaced by another. In these cases, the tester must manually make the correspondence between the removed element and the new one or confirm that, in fact, it was removed. At the end of his analysis, the tester confirms the changes that will be made in the tests for which his assistance was needed and they are repaired.

As in the assisted approach, the same kind of report is generated and the test suite is executed.

The rules used for the automatic analysis of the removed elements of each page are:

- First Rule:

  If there is one removed element and one new element with the same type and the same associated Label, a correspondence between them is established.

- Second Rule:

  If there is one removed element and one new element with different types and the same associated Label, a correspondence between them is established, only in case of the following transformations: Button with Link (and vice versa), Check Box with Radio Button (and vice versa) or Text Box with Select Box (and vice versa).

- Third Rule:

  If there is one removed element and one new element with the same type but different Associated Label, a correspondence between them is established, only if both are the only ones of that type that have not been matched.

- Fourth Rule:

  If there is one removed element but there are no new elements of the same type, or they are all already matched, the tool assumes that this element was, in fact, removed.

None of these rules ensures complete effectiveness or correction, however, one assumes the risk of applying them in order to decrease the effort required on the part of the tester in the process of maintaining the test scripts. In the end, all the modified test scripts are executed, and if the tester verifies that the changes made were not correct, he can undo them using the version control system. Furthermore, if for some reason, the tester doesn't want to take the risk of applying the automatic algorithm he can continue to use the assisted approach, where he is allowed to make the identification of removed and modified elements manually.

5 Evaluation

This section evaluates both approaches of our solution.

As a way to evaluate the automatic approach, it was used the test scripts previously recovered during the evaluation process of assisted approach, and others that had been developed since. As this evaluation took place simultaneously with the development of the NextWay platform, it was possible to take advantage of two situations that occurred during the project, where the programmers made changes to software that it broke its correspondence with some of the test scripts.

To analyse the effort required to perform the maintenance of test scripts, either manually or through the developed tool, we used the following metric to measure the maintenance effort:

- Manual removal of an action from a test script has a weight of 1;
- Manual modification of an action of a test script has a weight of 1;
- Manual removal of a test script from the test suite has a weight of 1;
- Manual identification (using the maintenance tool) of an element's transformation has a weight of 1;
- Manual confirmation (using the maintenance tool) of the removal of an element has a weight of 0,5 (It is considered this value because the tool automatically identifies this situation, the tester has only to confirm that the element was actually removed and not replaced by another);
- Removal of a test from the test suite (using the maintenance tool) has a weight of 0 (It is considered this value because this process is completely automatic and transparent to the tester.).

5.1 Evaluation of Assisted Approach

As a way to evaluate the implemented solution it was used a set of test scripts previously developed within the NextWay project and whose correspondence with the platform had been broken. Using the assisted approach, it was performed the recovery of those test scripts.

Early in the project, there was a set of 755 test scripts that were manually maintained for some

time but, due to the costs associated with this process, were ultimately abandoned.

After the maintenance tool was applied to these test scripts, it was possible to recover 460 of the initial set of which 37 failed in its implementation due to errors detected in the application under test. Of the 295 test scripts removed from the test suite, 115 exercised features that were no longer supported and 180 navigated through pages removed from the platform. All test scripts that were recovered had to be repaired by the maintenance tool.

To be possible to successfully recover 460 of the initial 755 test scripts, it was necessary for the tester to confirm the removal of 357 elements and manually identify the transformations undergone by 660 elements (making the correspondence between old and new elements). Based on this information, the maintenance tool automatically repaired the tests by removing 723 actions and modifying 2703 actions, which corresponds to the manual effort that would be required to the tester without the aid of the maintenance tool.

Using the accounting presented at the beginning of this section, the effort required to perform the maintenance of the 755 test scripts was:

- Manually: **3721** (295 scripts removed from the test suite, 723 actions removed, 2703 actions modified).
- Using the maintenance tool (assisted approach): **838,5** (335 * 0,5 confirmation of removed elements e 660 identification of element's transformation).

The number of actions necessary to repair the test scripts using the maintenance tool (in assisted mode) is smaller then the number of actions necessary in a manual approach. Using the tool in the assisted mode, the tester only has to identify each modification in the platform once and that information is used to repair the test scripts. In a manual approach, the test scripts have to be repaired one by one. If one modified element is referenced in many test scripts, the tester would have to manually repair each one, while with the maintenance tool the tester only has to identify that transformation once.

With these results we can say that there was a decrease of 77,5 % in the effort required to perform the maintenance of test scripts using the assisted approach of the maintenance tool compared with a completely manual process.

5.2 Evaluation of Automatic Approach

The automatic approach differs from the assisted one by the introduction of heuristics to identify the changes in the graphical interface of the platform between the two versions under consideration. The main goal is to reduce and ideally eliminate the manual effort required on the part of the tester in this process.

To evaluate this approach it was used a different context. Using the test scripts, previously recovered during the first evaluation process and others that had been developed since, two real situations of the evolution of the platform were analysed. In both scenarios the programming team made changes to the software that broke its correspondence with some of the existing test scripts.

In both scenarios, first it was applied the assisted approach and subsequently the automatic approach, measuring how the tester's effort was reduced from the first to the second.

Scenario A

Manual Effort	Effort Assisted Ap.	Effort Automatic Ap.
89	35	23

Table 1: Effort of the tester in Scenario A, with the different approaches

With the results shown in Table 1, we can say there was a decrease of 60,7% in the effort required to perform the maintenance of test scripts using the assisted approach of the maintenance tool compared with a completely manual process.

Using the tool in automatic mode the tester's effort was 34,3% less than in assisted mode and 74,2% less than in a completely manual process.

Scenario B

Manual Effort	Effort Assisted Ap.	Effort Automatic Ap.
73	52	3

Table 2: Effort of the tester in Scenario B, with the different approaches

In the second scenario, the improvement of the automatic approach was much better. Table 2 shows the effort of a tester for different approached. We can say there was a decrease of 28,8% in the effort required to perform the maintenance of test scripts using the assisted approach of the maintenance tool compared with a completely manual process.

Using the tool in automatic mode the tester's effort was 94,2% less than in assisted mode and 95,9% less than in a completely manual process.

The described scenarios had different results because they are related to two distinct situations

where the evolution of the graphical interface of the platform was completely different. The automatic approach can make the recovery more or less test scripts, without the intervention of the tester, depending on the changes undergone by the platform.

6 Final Remarks

During the NextWay project, the problem of the maintenance of test scripts was strongly felt by the testers, which motivated the development of the solution presented in this paper.

It was implemented a tool for the maintenance of the test scripts that must be used when changes in the GUI of the NextWay platform break its correspondence with the test scripts. Two approaches can be used by the tester, the assisted ot the automatic one. In general, with both approaches, the effort of the tester in the process of repairing test scripts was greatly reduced.

As future work we intend to make the maintenance process increasingly automated and less dependent on the tester. To this end we intend to evolve the solution in order to:

- Make the automatic approach increasingly complete, covering more cases where it is possible to identify whether an element was in fact removed or it has changed.
- Support a larger number of types of transformations.
- Give more support to the tester on the analysis of the execution of the test suite after making repairs on its test scripts. The tool should go through the list of failed tests and provide the tester a list of those which were previously repaired (using the automatic or the assisted approach).

References

[1] I. S. T. Q. Board. *Certified Tester Foundation Level Syllabus*. ISTQB, 2010.

[2] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso. Water: Web application test repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, ETSE '11, pages 24–29, New York, NY, USA, 2011. ACM.

[3] B. Daniel, V. Jagannath, D. Dig, and D. Marinov. Reassert: Suggesting repairs for broken unit tests. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 433–444, Washington, DC, USA, 2009. IEEE Computer Society.

[4] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, and M. Pezzè. Automated gui refactoring and test script repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, ETSE '11, pages 38–41, New York, NY, USA, 2011. ACM.

[5] M. Grechanik, Q. Xie, C. F. Chen, and C. Fu. Guide: A gui differentiator. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 395–396, Washington, DC, USA, 2009. IEEE Computer Society.

[6] M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving gui-directed test scripts. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 408–418, Washington, DC, USA, 2009. IEEE Computer Society.

[7] L. G. Hayes. Evolution of automated software testing. *Automated Software Testing*, pages 14–20, Aug. 2009.

[8] D. Johnson. Is your automation agil or fragil? *Automated Software Testing*, pages 22–27, May 2009.

[9] D. Johnson. 2nd annual ati automation honors. *Automated Software Testing*, page 14, Nov. 2010.

[10] K. Li and M. Wu. *Effective Gui Test Automation*, chapter 1–2. Sybex, 2005.