Automated Regression Testing of BLAISE INTERNET: A Case Study

Karen Brenner and Sudha Manoj, Westat

A commercial off the shelf (COTS) testing product, TestPartner, is being used to automate test cases for Blaise Internet projects. This paper will review the pros and cons of automated testing and present considerations in making a decision about whether automation is the best option for a particular project. We will look at how we implemented automated testing and discuss issues that Blaise Internet presented and how they were resolved.

Automated Testing Lifecycle

This paper focuses on Automated Testing for comprehensive regression testing changes in a Blaise Internet instrument. Like the overall software development lifecycle (SDLC), test automation has its own lifecycle consisting of the following six steps:

1. Level of Effort Analysis: The decision whether to automate testing is based on analyzing the field length of the project, system stability, available resources, test environment, tool compatibility, and other factors. A number of significant benefits and false expectations of automated testing are discussed later in this paper.

2. Test Tool Acquisition: A test tool can be developed or purchased after comparing and evaluating tools on the market. TestPartner (by MicroFocus) is the standard automation tool currently used by the Westat Testing Unit; it has been successfully implemented on Westat projects.

3. Automated Testing Introduction Process: An overall test process and strategy needs to be in place before introducing automated testing for any new project. For example, documented system changes must be passed to testing to update the version of suites of scripts and expected results.

4. Test Planning, Design, and Development: This phase includes setting up a preliminary schedule and test environment guidelines, and developing the automated scripts. The Testing Team leads this phase of the work with staff equipped with a second desktop PC for developing and tuning the automated scripts. This phase includes close collaboration with the developer and/or tool tech support to help manage how the tool integrates with the system.

5. Execution of Tests: The test environment needs to be in place as planned. The automated scripts must have dedicated PCs on which to run. Dedicated PCs for automated testing are totally independent of development PCs, so that they can provide "unpolluted" platforms on which to test. Thus, they should be as close to the end user configuration as possible. It may be desirable for testing platforms to have increased processing speed and memory. This consideration has to be weighed against testing on a system that is identical to the end user system. The tester will execute automated scripts and provide test results for evaluation.

6. Test Program Review and Assessment: Test program review and assessment activities need to be conducted throughout the testing lifecycle, to allow for maintenance of scripts and continuous process improvement activities.

Significant Benefits of Test Automation

Once it has been determined that automation is appropriate for a project, the team can look forward to the following benefits:

1. Increased depth and breadth of regression testing.
2. Elimination of long, repeatable manual test cases. (Although automation doesn't eliminate manual testing, it does replace lengthy repeatable test cases so that testers can focus on particular issues.)
3. Reduction of the schedule.
4. Unattended testing. Automated tests can run unattended and overnight.
5. Improved quality of the test effort.
6. Improved assessment of system performance. Besides allowing for greater consistency and test coverage, automation can be used to ensure that the system's performance requirements meet or exceed user expectations.

False Expectations for Automated Testing

After considering the automated testing life cycle, the benefits of automation must be weighed against false expectations:

Automation can be implemented at any time

Automated testing requires a long-term project with an instrument that is relatively stable, that needs to have run a specified set of regression tests on a regular basis that have predictable results. If the project tries to implement automation prematurely, while a system is still being developed, it will increase the maintenance required to keep suites of scripts running that can provide useful feedback about software issues.

Automation can replace manual testing

Automated test tools should be viewed as *enhancements to manual testing*; they will not develop a test plan, design and create test cases and procedures, and execute the tests. The test team will never achieve 100% test automation of an application, because it is not possible to test all combinations and permutations of all inputs. An experienced tester will certainly need to execute tests during exploratory testing that were not predicted when writing test cases.

Automation is easy

Vendors sell an automated tool by exaggerating its ease of use, usually referring to it as a "record/playback" tool. Automation is actually more complicated than that, because recorded test scripts must be enhanced manually with code to make the scripts robust, reusable, and maintainable. To be able to modify the scripts, the tester must be trained and become an expert on the tool's built-in scripting language.

One tool does it all

Currently, one single test tool will not fulfill all testing requirements for most projects. Different tools are used for regression, file comparison, load, and other aspects of testing.

Immediate test effort reduction

In addition to the learning curve associated with applying automated testing to a project, test automation requires that the team pay careful attention to automated test procedure design and development. The automated test effort can be viewed as its own software development life cycle, complete with the planning and coordination issues that come along with a development effort. Time is invested up front to organize, plan, script, and debug.

Selecting an Automated Tool

There are many tools in the market which support automation. TestPartner is the testing tool we selected to automate our regression testing process at Westat. Like most automation test tools, you can create scripts automatically using TestPartner's record facility. When recording your actions, the responses of the application you work with are translated into TestPartner scripts. TestPartner can record just keystrokes, mouse moves and clicks. It works best at an object level –- identifying objects by name. Your actions are translated into simple commands. TestPartner lets you quickly record and execute test scripts. The tester can then modify test scripts to include hand-coded programming that cannot be recorded, to enhance the scripts and make them easier to maintain. All automated test tools will require this hand-coding to make scripts robust, even though tools are generally marketed as "record and playback."

In a basic out-of-the-box Blaise for Windows script, TestPartner is not able to uniquely identify the objects on the screen, since Blaise for Windows does not supply a unique attribute to the windows control. One option we've implemented with a Blaise for Windows application is to build a web front end for a Blaise survey, that uses the Blaise database, and then easily build automated test scripts using the HTML web interface.

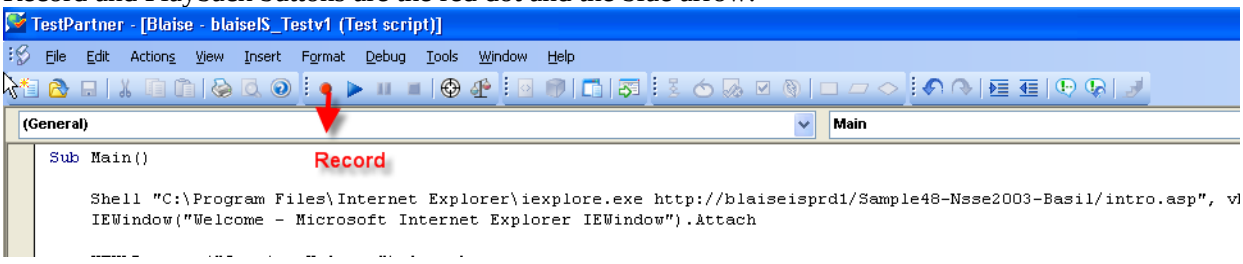
TestPartner works well with Blaise Internet and raises very few issues.

Example of how the tool works

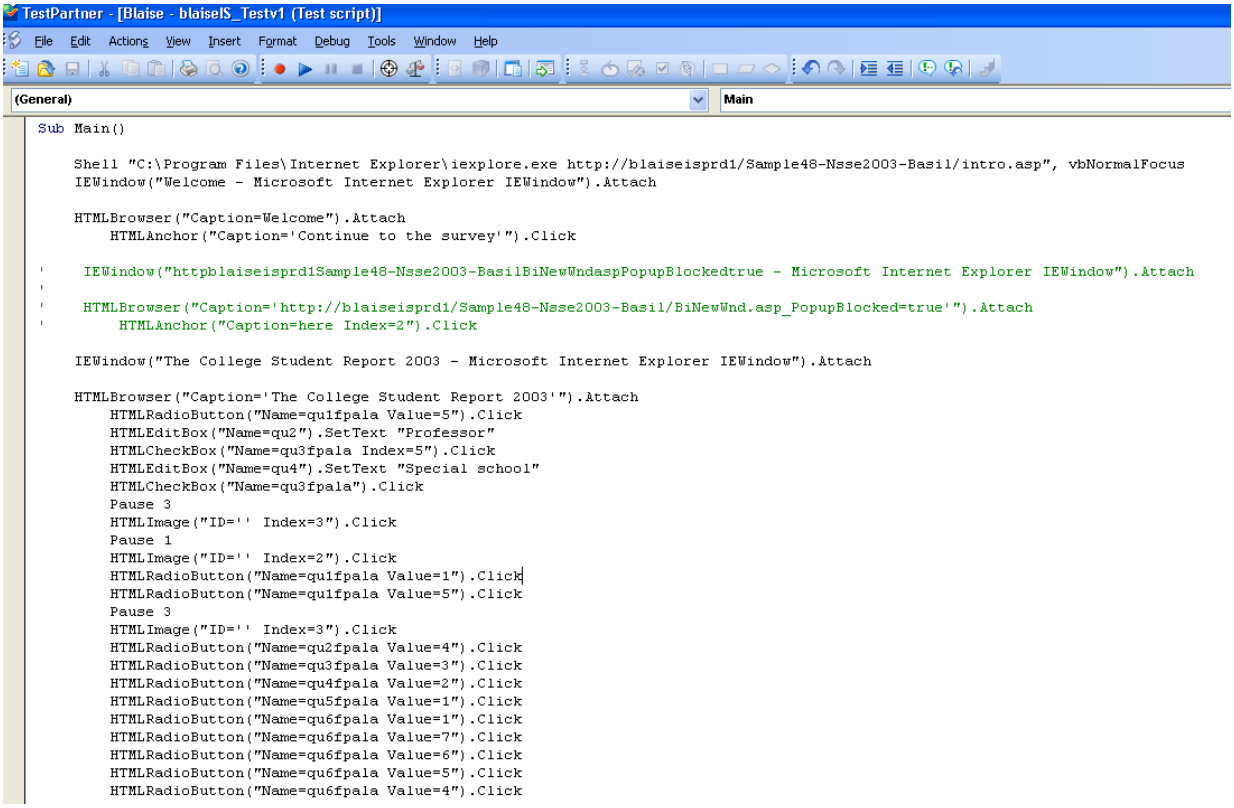
For demonstration purposes, we have version 1 of an instrument which is relatively small and has already been determined to be a candidate for automation. First, we need to record the script by going through the instrument manually. Each object in the instrument has an object name with unique properties. The object name is displayed in the script when the tool generates the code. The script generating options we use record the tester's actions such as keystrokes and mouse clicks as they are applied to the objects on the screen. These options make the script more understandable when read and allow us to more easily maintain it. Once the script is recorded the first time, we can play it back. After the script runs, the results will be displayed in the results window. If the script runs through correctly, the results show as passed. If the instrument has any changes or does not run, the script fails and the results are displayed as failed.

Here is an example of a script running on version 1 of the instrument.

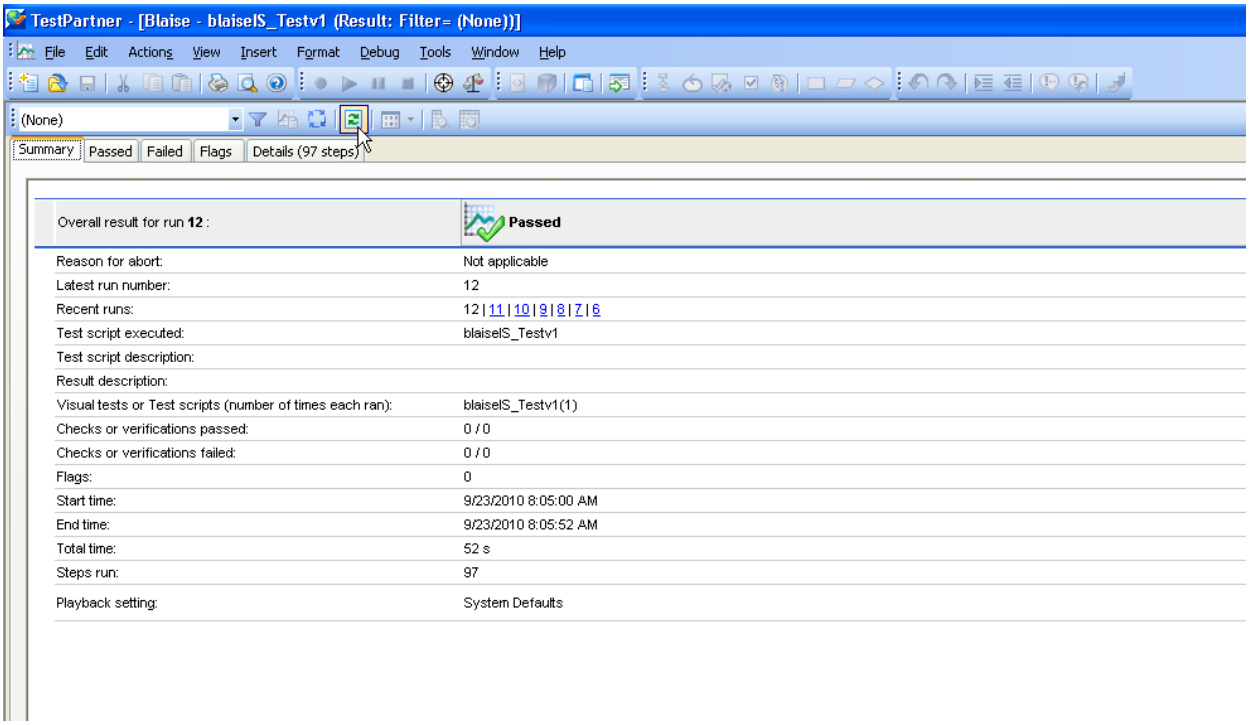
Record and Playback buttons are the red dot and the blue arrow:



Script:



Results Window:



If a script stops during playback, we know that there is an issue. When it is determined to be a defect, we enter the issue in our standard bug tracker. If it is not a defect, for example an intentional change to the instrument, we have to modify the script to make it run. This requires maintenance of the scripts as the versions of the instrument keep changing. It is not very difficult, but scripts have to be updated on an ongoing basis. One can edit the code manually to comply with the screen or re-record portions of scripts to incorporate the new information and allow the script to continue running.

First, it is important to note that automation tools have a variety of checks that can be used for more test coverage. Without checks, you're only testing navigation. Adding checks lets you confirm that the application is validating input, performing calculations correctly, saving data reliably, and reporting accurately, by comparing actual responses to expected responses and reporting any discrepancies.

Here are some of the checks that can be used in TestPartner.

Bitmap Checks: These checks compare a bitmap in the target application with one that was previously defined. Bitmap checks are used to verify the appearance of toolbars, the desktop, and other windows that contain non-textual information.

Content Checks: A content check enables you to verify the contents of controls that the tool supports. Currently, tables and list controls in a Windows-based or Web-based application are supported. List controls include list boxes, drop-down lists, combo boxes, and various "tree" type controls, such as those used in Windows File Manager and Windows Explorer.

Field Checks: Field checks enable you to conduct specific types of text comparisons, including date and time, of individual fields you select in a window or area.

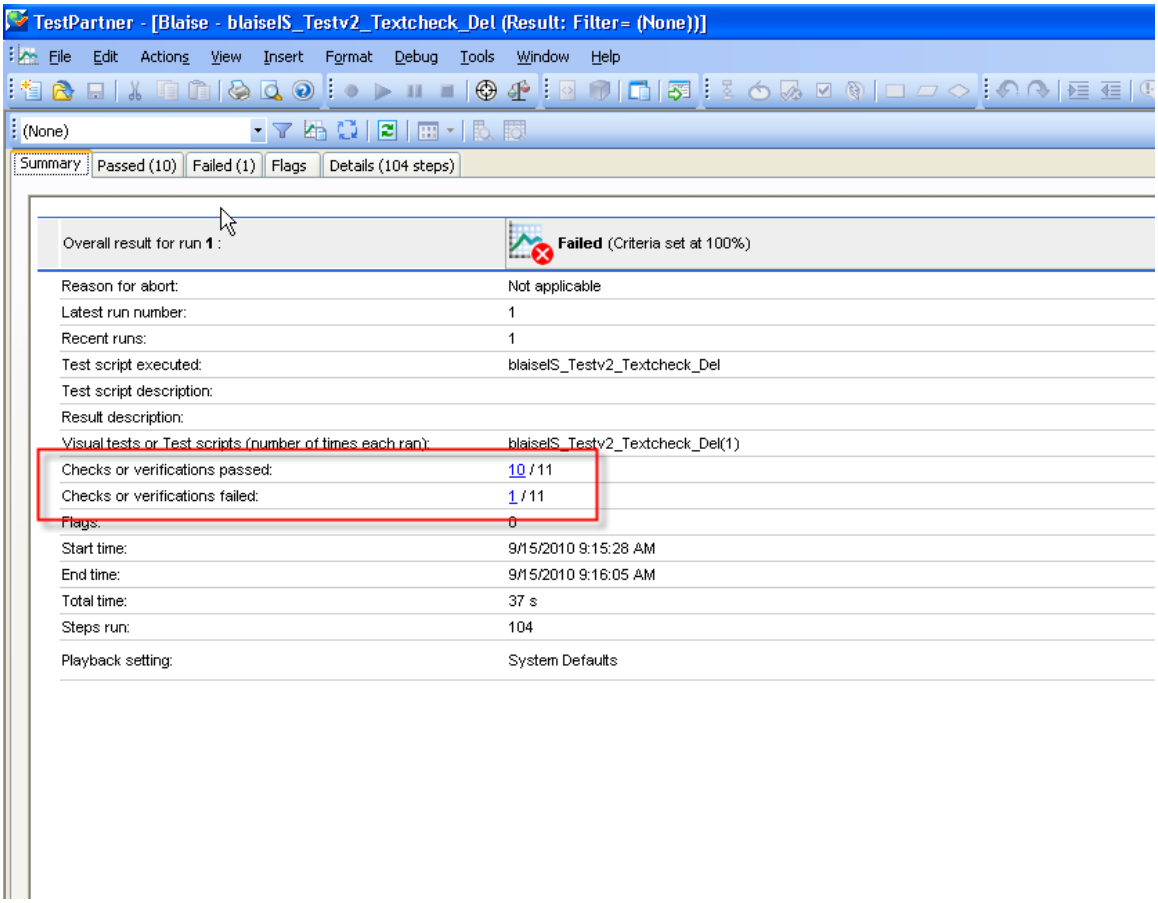
Text Checks: These checks can be used to verify the text on a screen. You can check an entire window and mask the area(s) that contain variable data, such as the current date or anything else that you want to ignore during the check. One can add a check by simply selecting the area of the window that contains the text to capture.

In the following examples, we will be examining an instrument where a field has been added to the page and a response list has been modified. We want to look at how to set up the script to spot various types of changes and how to maintain the scripts.

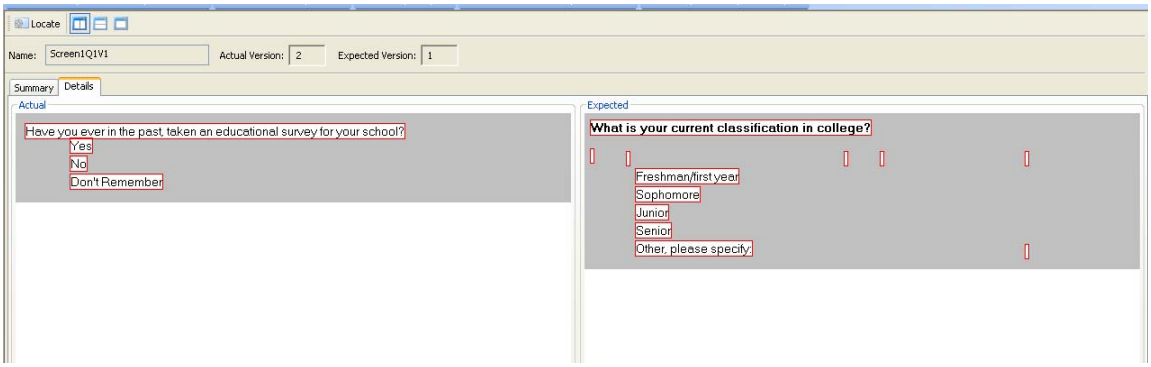
Text Check Example

In this example, version 1 of the instrument asks a Yes or No question on screen1, while version 2 of the instrument lists classifications in college. We captured the text and ran the same check on version 2. The script is expecting to find certain text in a specific location, and notices that there is a discrepancy. Since the text does not match, the check fails and is displayed on the results screen.

Failure Screen:



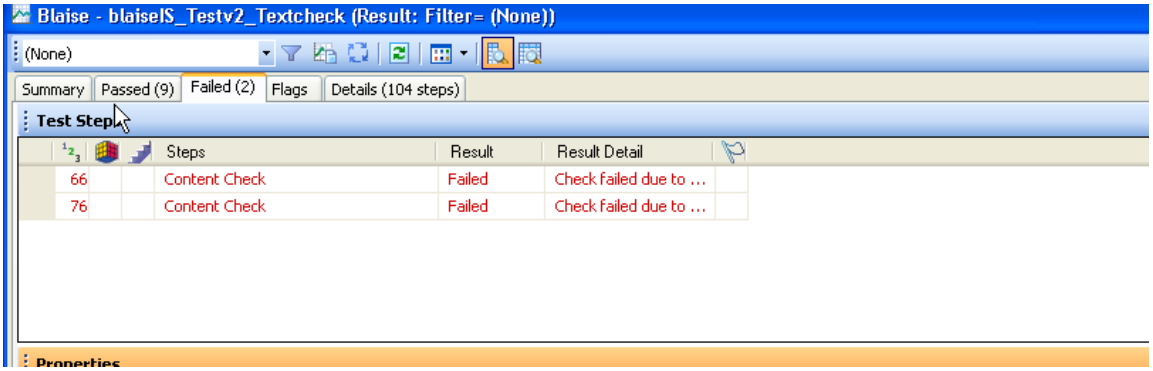
When you click on the failed check, it takes you to the details page, and the differences of the text are displayed as shown in the screenshot below.



Content Check Example

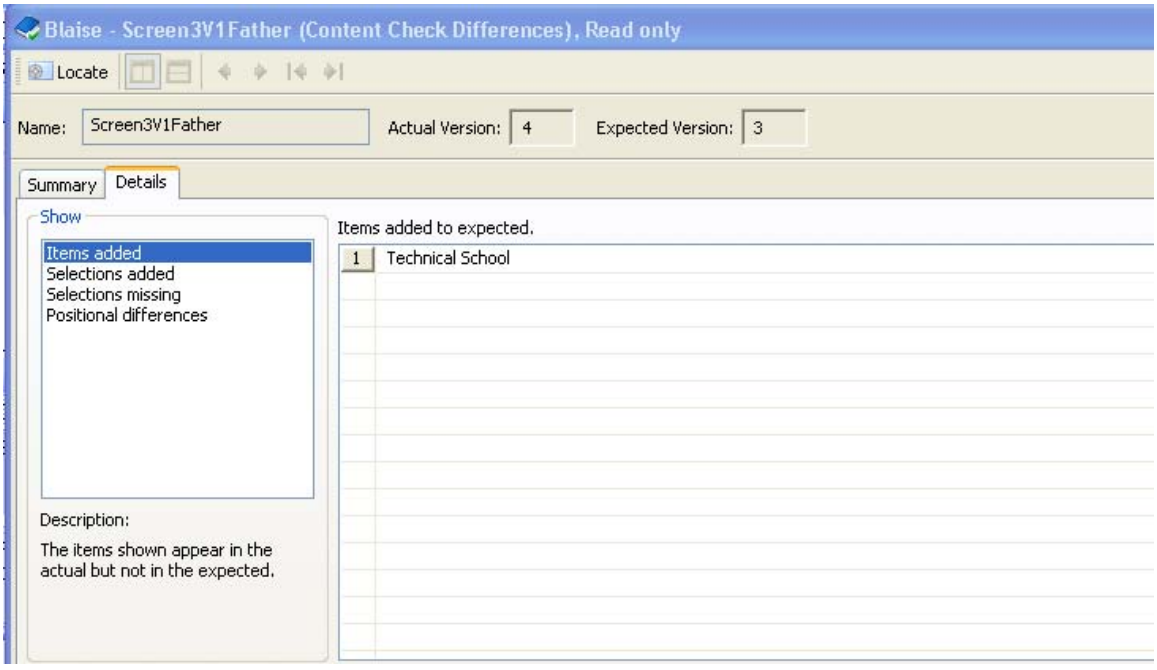
Content checks can be used to check the text of the items in a list or the number of items in a list, and if a list item is selected, the content check can be used to check the number of columns and rows in the table. One can add a check by simply selecting the field with the dropdown list to capture the items in the list.

In this example, version 1 of the instrument has eight items in the dropdown list on screen3, while version 2 of the instrument has been modified to have nine items. We created a content check in version 1 and ran the same check with version 2. Since the content of the items in the dropdown does not match, the check fails and it is displayed on the results screen.

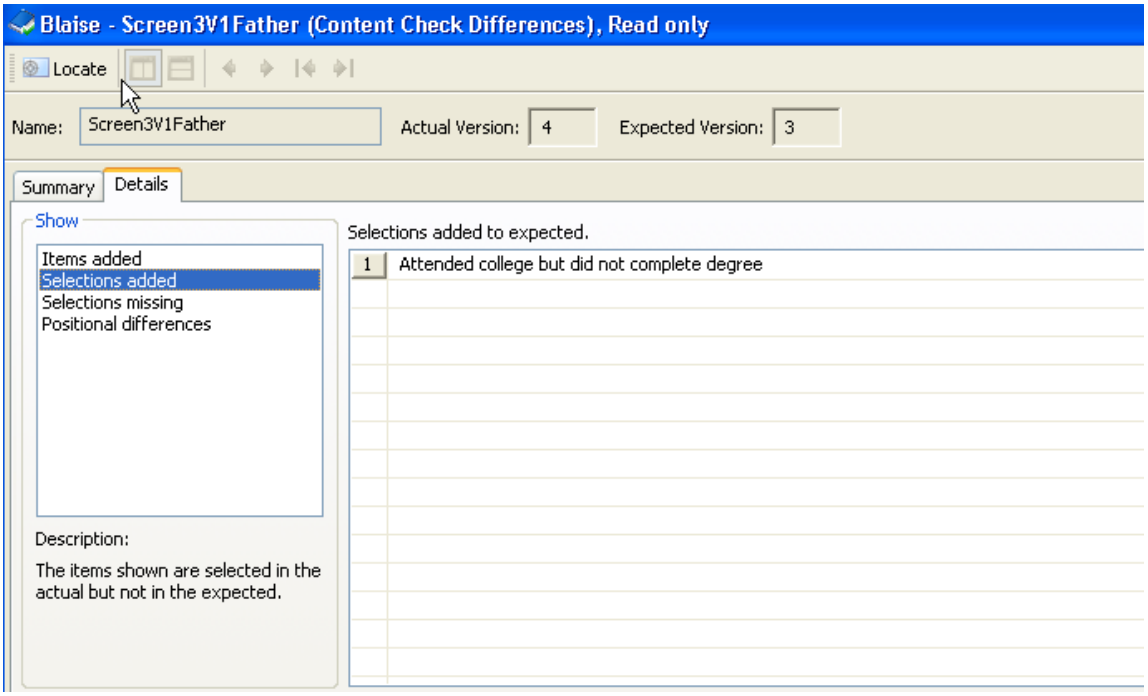


When you click on the failed check, it takes you to the details page and the differences in the content are displayed as shown in the four screenshots below. In this case, it shows any items added, selections added, selections missing, and any positional differences.

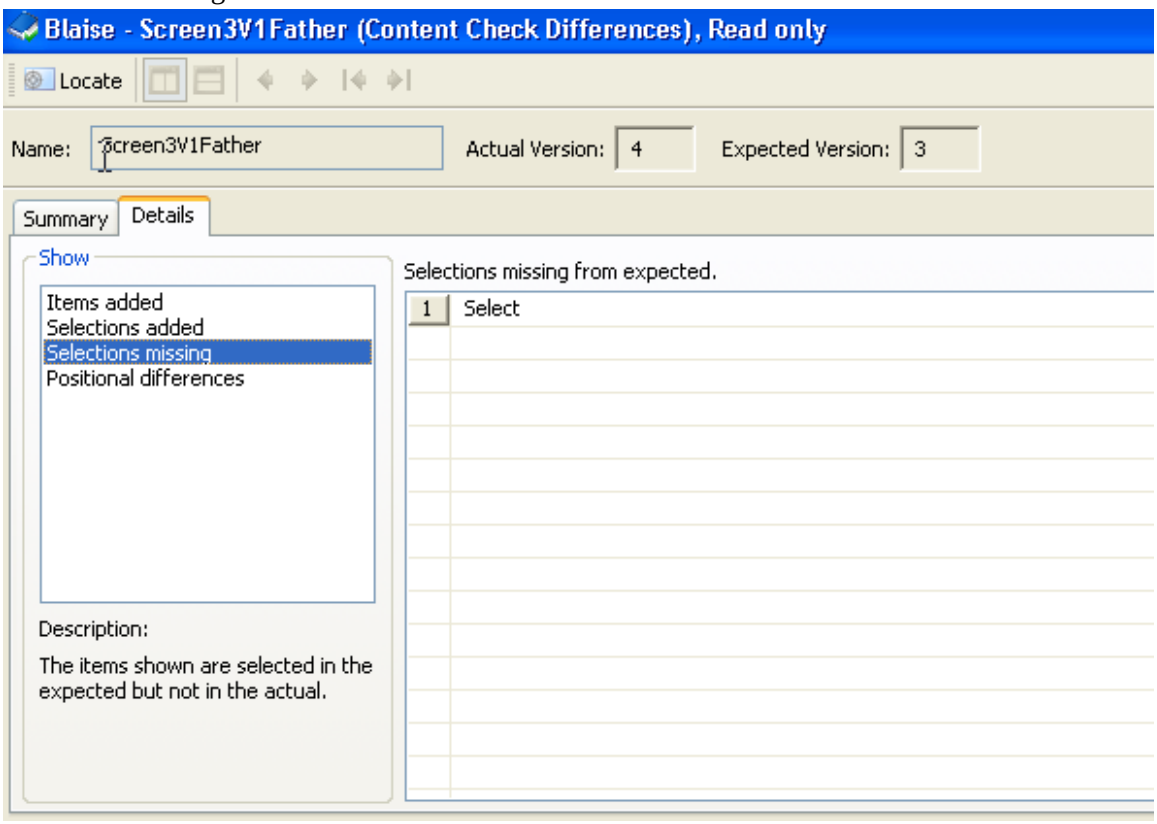
Items added:



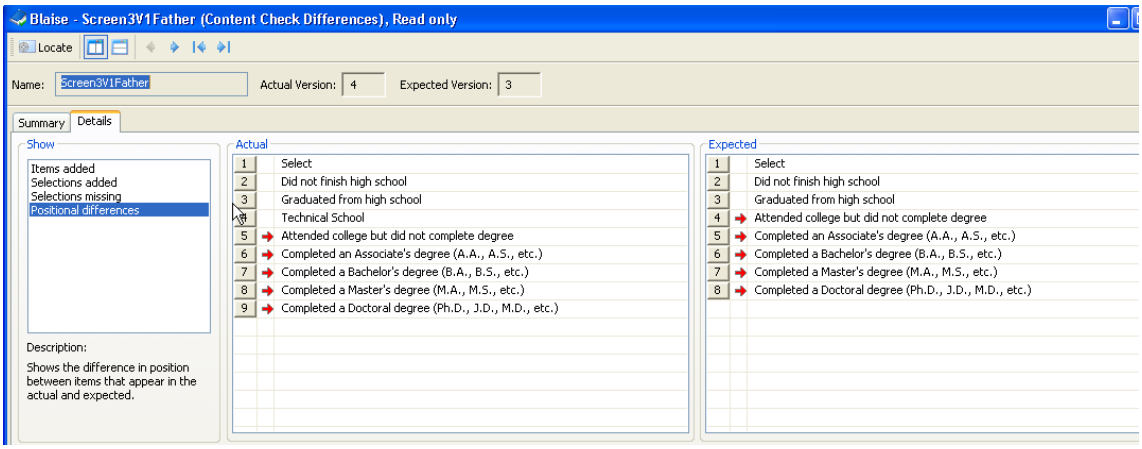
Selections added:



Selections Missing:



Positional differences:

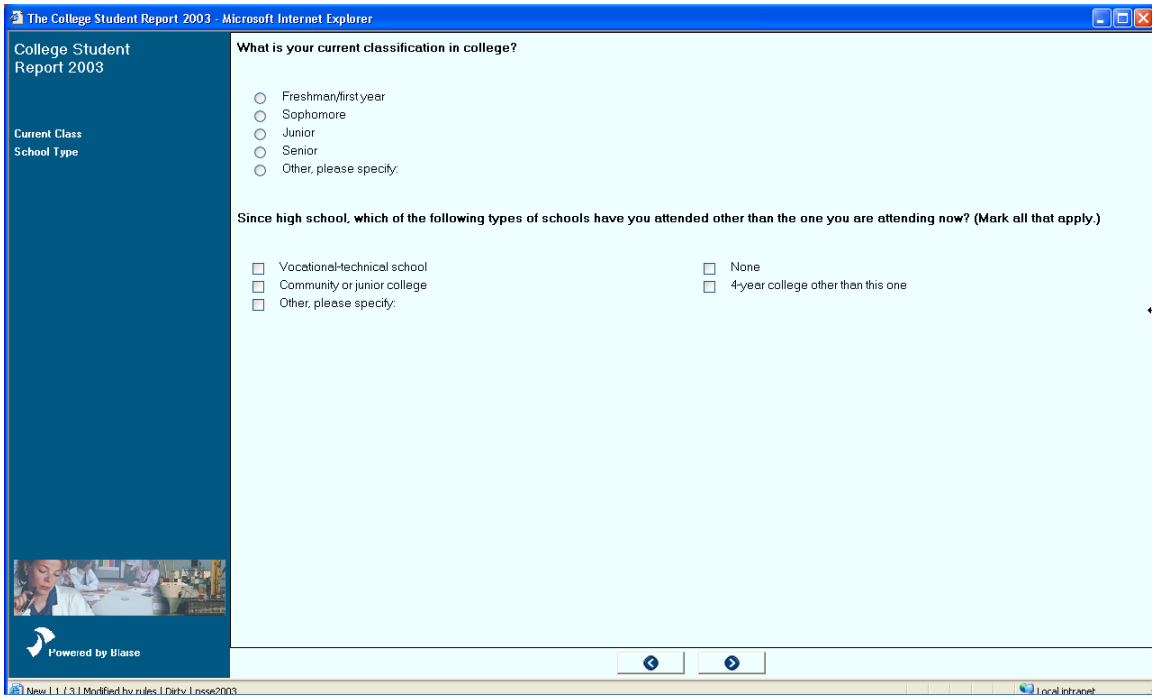


A script without checks would still run without errors as long as the navigation remained the same as originally scripted.

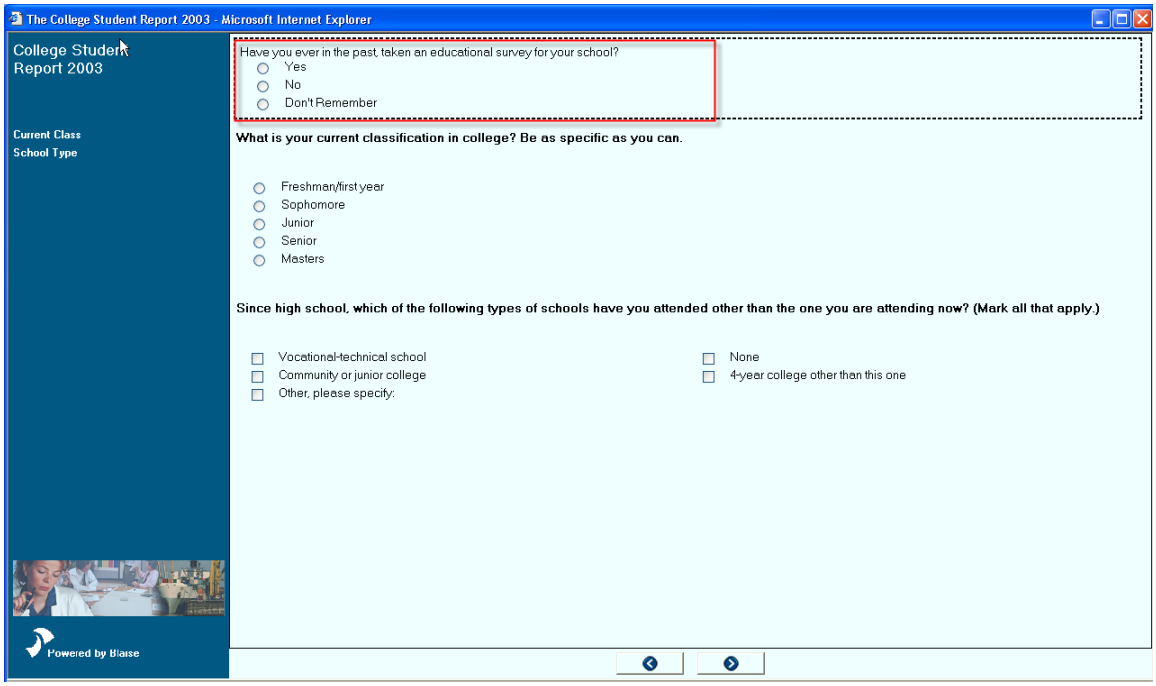
Example of an Issue with Script Maintenance – New Objects

The one area where we had additional maintenance issues using our tool with Blaise Internet was when there was a question (object) added to the instrument.

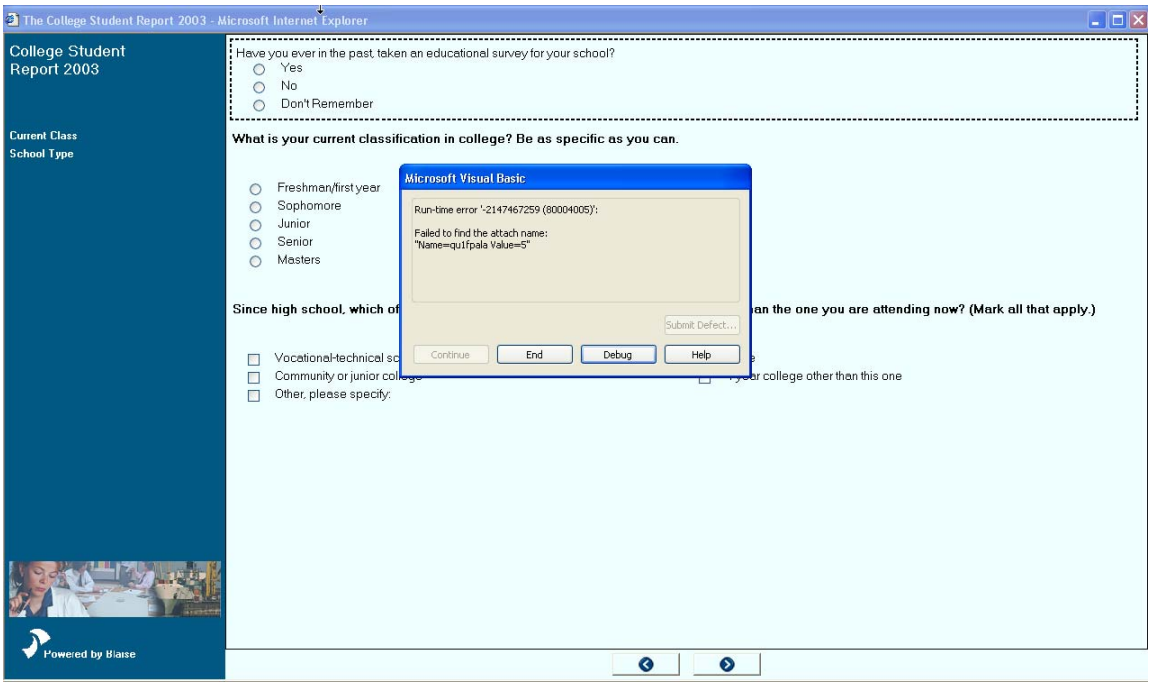
On this screen which is version 1, there are two questions.



The new version 2 has three questions. The first question seen in the red box has been added.



When we run the script during regression testing, the script stops when it gets to the new object. The script is expecting to be located on the classification field, but instead the focus is on the newly added survey field. Once it stops a debug screen appears as shown below.



When we click on Debug, it opens up the script and highlights the line where the problem exists, as shown in the screenshot below:

```
    IEWindow("Welcome - Microsoft Internet Explorer IEWindow").Attach

    HTMLBrowser("Caption=Welcome").Attach
        HTMLAnchor("Caption='Continue to the survey'").Click

'    IEWindow("httpblaiseisprd1Sample48-Nsse2003-BasilBiNewWndaspPopupBlockedtrue - Microsoft
'
'    HTMLBrowser("Caption='http://blaiseisprd1/Sample48-Nsse2003-Basil/BiNewWnd.asp_PopupBloc
'        HTMLAnchor("Caption=here Index=2").Click

    IEWindow("The College Student Report 2003 - Microsoft Internet Explorer IEWindow").Attach

    HTMLBrowser("Caption='The College Student Report 2003'").Attach
        HTMLRadioButton("Name=qu1fpala Value=5").Click
        HTMLEditBox("Name=qu2").SetText "Professor"
        HTMLCheckBox("Name=qu3fpala Index=5").Click
        HTMLEditBox("Name=qu4").SetText "Special school"
        HTMLCheckBox("Name=qu3fpala").Click
        Pause 3
        HTMLImage("ID='' Index=3").Click
```

The result is that when an object is changed (added or removed) on a Blaise Internet page, the properties of all pre-existing objects on that page also change. One can edit or re-record the script to add the missing question or edit the code manually to correspond with the new screen updates and continue the script.

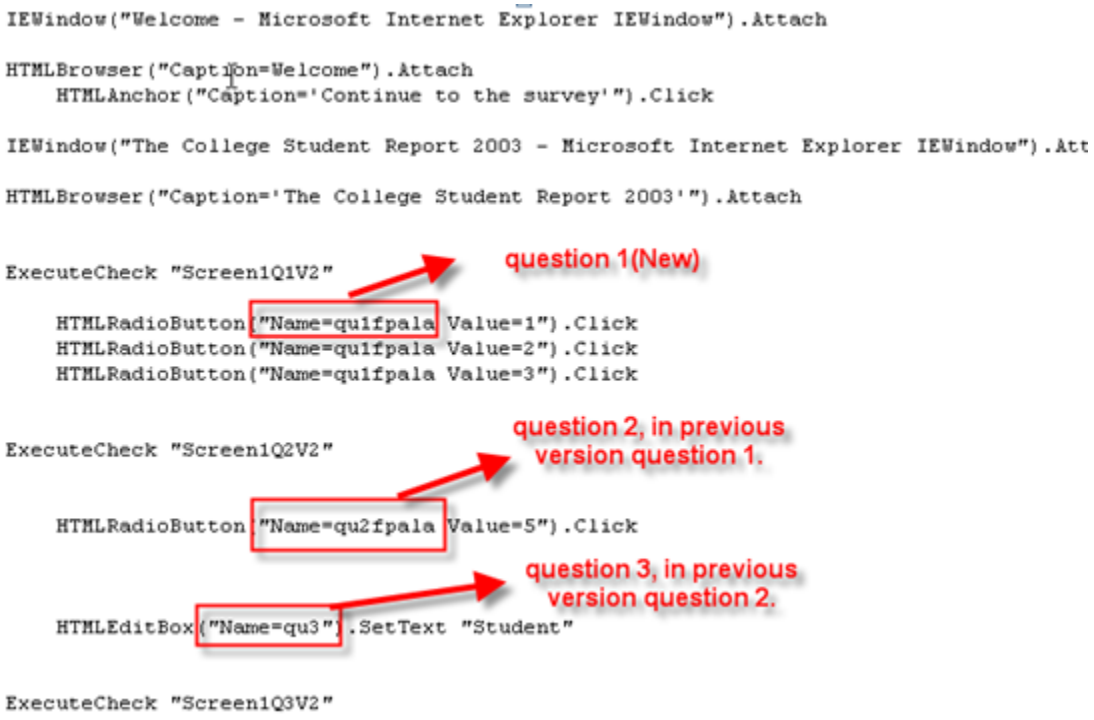
In our example, the script has the object name for question 1 as “Name=qu1fpala”, and the object name for question 2 as “Name=qu2” as seen in the screenshot below.

```
ExecuteCheck "Screen1Q1V1"


HTMLBrowser("Caption='The College Student Report 2003'").Attach
    HTMLRadioButton("Name=qu1fpala Value=5").Click

    HTMLEditBox("Name=qu2").SetText "Professor"

ExecuteCheck "Scr1Q2V1"

    HTMLCheckBox("Name=qu3fpala Index=5").Click
    HTMLEditBox("Name=qu4").SetText "Special school"
    HTMLCheckBox("Name=qu3fpala").Click
    Pause 3
    HTMLImage("ID='' Index=3").Click
    Pause 1
    HTMLImage("ID='' Index=2").Click
    HTMLRadioButton("Name=qu1fpala Value=1").Click
```

question 1

question 2

After the new question is added, the properties of all pre-existing objects on that page also change. Now you will notice that the object name for question 2 is now “Name=qu2fpala” instead of “Name=qu1fpala”.



This indicates that the object properties change on that particular page when objects are added or removed. Therefore, the script has to be re-recorded for that section or the user can manually change the code. The maintenance issue is made easier because in general there are not many objects on a Blaise Internet page.

Conclusion

In summary, there are many automated testing tools. This paper addresses our experience using one COTS product. We were able to use an automated testing tool to successfully perform regression tests on a Blaise Internet instrument. The regression testing is ongoing and will be considered for other complex Blaise instruments.