Spot the Difference:
Automated Visual Regression Testing with Blaisium

Angelo Pascale, Joris Bleus and Arthur van den Berg, Statistics Netherlands

1. Abstract

With each new Blaise release, not only should the functionality of existing Blaise web questionnaires remain the same, they should also still look the same. It takes a lot of time to test this manually. That is why we want to test this automatically. There are no affordable, easy-to-use tools in the market yet that can do this for all operating systems and devices. Therefore we have built a Python framework that can do this: Blaisium. With Blaisium, one can test web questionnaires on different devices and device farms. At the Blaise test team we have linked Blaisium to Browserstack. In this way we can test the visual aspects of web questionnaires on Windows, macOS, Android and iOS devices.

2. Automated Visual Regression Testing

2.1 Regression

Regression is the phenomenon that software features that worked well before, do not work properly anymore after the code of the software has been modified. It is not uncommon to unintentionally break something while fixing something else.

2.2 Regression Testing

Regression testing is the act of re-running previously executed software tests after the code of the software has been modified. The purpose of regression testing is to verify that the modification of the code does not have unforeseen consequences.

2.3 Visual Regression Testing

Visual regression testing is testing for unintended visual changes due to modifications of the code. A visually flawed page might still function correctly and might still pass all the functional regression tests. Therefore it is important to explicitly test for visual regression.

The basic idea behind visual regression testing is simple. You have an image of what your user interface looked like before the code modification and you compare it with what your user interface looks like after the code modification. If the images match, the test passes. If the images do not match, the test fails and you either need to repair the code or update the reference image.

2.4 Automated Visual Regression Testing

Manual regression testing in general and manual visual regression testing in particular, is very time-consuming, tedious and error prone. That is why it is a good idea to automate your visual regression tests. Automated tests are more reliable than manual tests. They can be executed in exactly the same way every time and can detect differences that might be overlooked by humans. They can also be executed much faster. Once created, automated tests can be run over and over again at almost no additional costs.

3. Challenges of Automated Visual Regression Testing

Regression testing in general and automated visual regression testing in particular present us with some challenges.

3.1 Ensuring adequate coverage

The visual aspects of your web pages should look exactly right on all common devices, operating systems and browsers. There are hundreds of possible combinations of devices, operating systems and browsers and they might all render the same page a little differently. Therefore it is necessary to execute the tests on a set of device - operating system - browser combinations that offers good coverage of the possible combinations that users might actually use.

3.2 Testing on real devices

Simulators and emulators never work exactly the same as the actual devices they try to mimic, so we should test on real devices. But since there are so many device - operating system - browser combinations that should be tested, it would be very time-consuming and expensive to purchase, set up and maintain all these devices.

3.3 Minimizing the lead time of test execution

Because we need to test a lot of visual properties and controls on a lot of device - operating system - browser combinations, this will take a lot of time. There may not even be enough time to run the full visual regression test for every build. We have to find a way to reduce the lead time of test execution.

3.4 Tuning the strictness of comparison

If our tests verify whether a survey page looks exactly the same as in a previous build (which means that not one pixel may be different), then a minor change, which cannot even be noticed with the naked eye and which is harmless, will cause these tests to fail. In visual regression testing we therefore prefer not to check whether the images match the reference images perfectly, but whether they correspond to a certain extent. However, this leaves us with the problem of how lenient we should be. Because if we are too lenient, harmful unintended visual changes might slip through. But if we are too strict, the regression tests might fail wrongly and we might have to replace the reference images unnecessarily.

3.5 Maintenance due to changing ID, XPath or other locators

Sometimes IDs and XPaths of visual objects are updated. Most test automation tools identify objects by their ID or XPath. To ensure that the tests containing these controls continue to work, we need to adjust those IDs and XPaths in each test in which they occur. This can be very time-consuming.

4. How Blaisium overcomes these challenges

Blaisium is a Python Test Automation Framework that makes use of Selenium, Appium and other useful libraries. The goal is testing web, mobile and desktop applications in a single project Blaise. Blaisium provides a simple way of choosing configured drivers, implements a page object pattern and includes a simple visual testing solution.

4.1 Ensuring adequate coverage

Blaisium uses BrowserStack to ensure an adequate device coverage. BrowserStack is a cloud web and mobile testing platform. It provides manual and automated testing of websites and mobile apps with a coverage of 2000+ real devices and browsers.

4.2 Testing on real devices

BrowserStack makes it possible for us to test on real devices without us having to purchase, install or maintain them. BrowserStack works out of the box with zero setup and zero maintenance.

4.3 Minimizing the lead time of test execution

The Blaisium Test Framework uses BrowserStack and configures the devices through a json file. It is easy and takes no setup time. Depending on our price plan we can easily run the same test on multiple devices up to 10 parallel tests to reduce the lead time of test execution.

4.4 Tuning the strictness of comparison

Visual testing in Blaisium is a tool for checking automatically if the survey is displayed as expected. The library that is used to make the comparisons of the images is called OpenCV. For the strictness of comparison, Blaisium uses Structural Similarity Index (SSIM) method. It actually measures the perceptual difference between two comparable images. It cannot judge which of the two is better. Blaisium has the knowledge to pass or fail the image due to the similarity value you give as parameter.

Blaisium workflow:

- It takes a screenshot of the current web page or mobile screen.
- If there is a previous image with the same name, it compares both of them and shows their differences. The current image is saved in a typical folder structure so you can easily reuse the image as baseline.
- If there is no previous image, it saves the current image to be used as a baseline. These images may be reviewed manually to assure that they can be used as expected baseline images.

After every execution, the tool generates a report image. See figure 1 for comparing two images and figure 2 for template matching.

In figure 1 you see the images converted to grayscale with the calculated structural similarity index. The ‘diff’ image contains the actual image differences between the two input images that we wish to visualize. The ‘thresh’ image is done on the ‘diff’ image to create a binary image. This binary image consists of pixels that can have one of exactly two colors, usually black and white.

Figure 1 - compare if the current screenshot image is the same as the baseline screenshot with a pass SSIM of 1

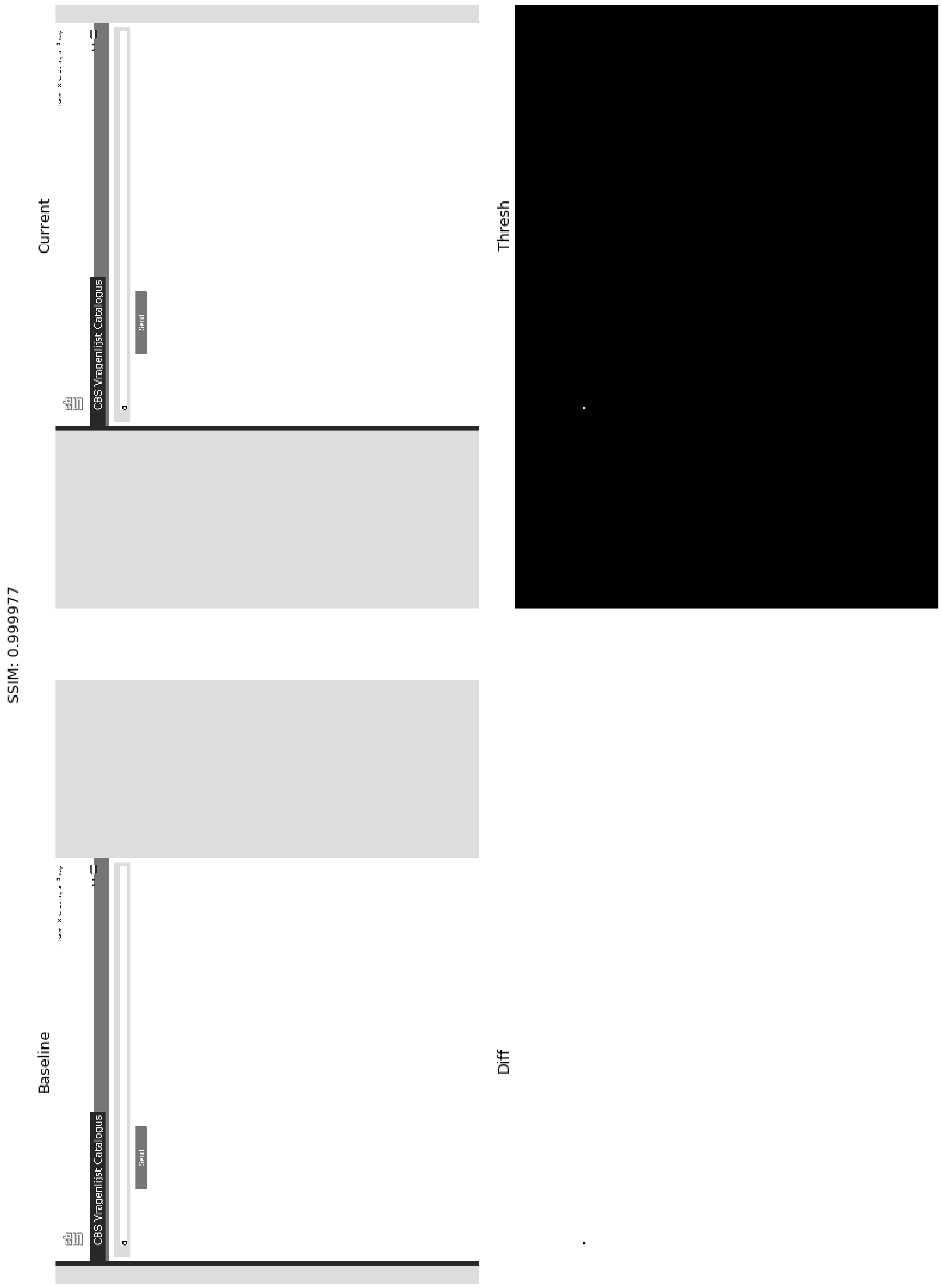
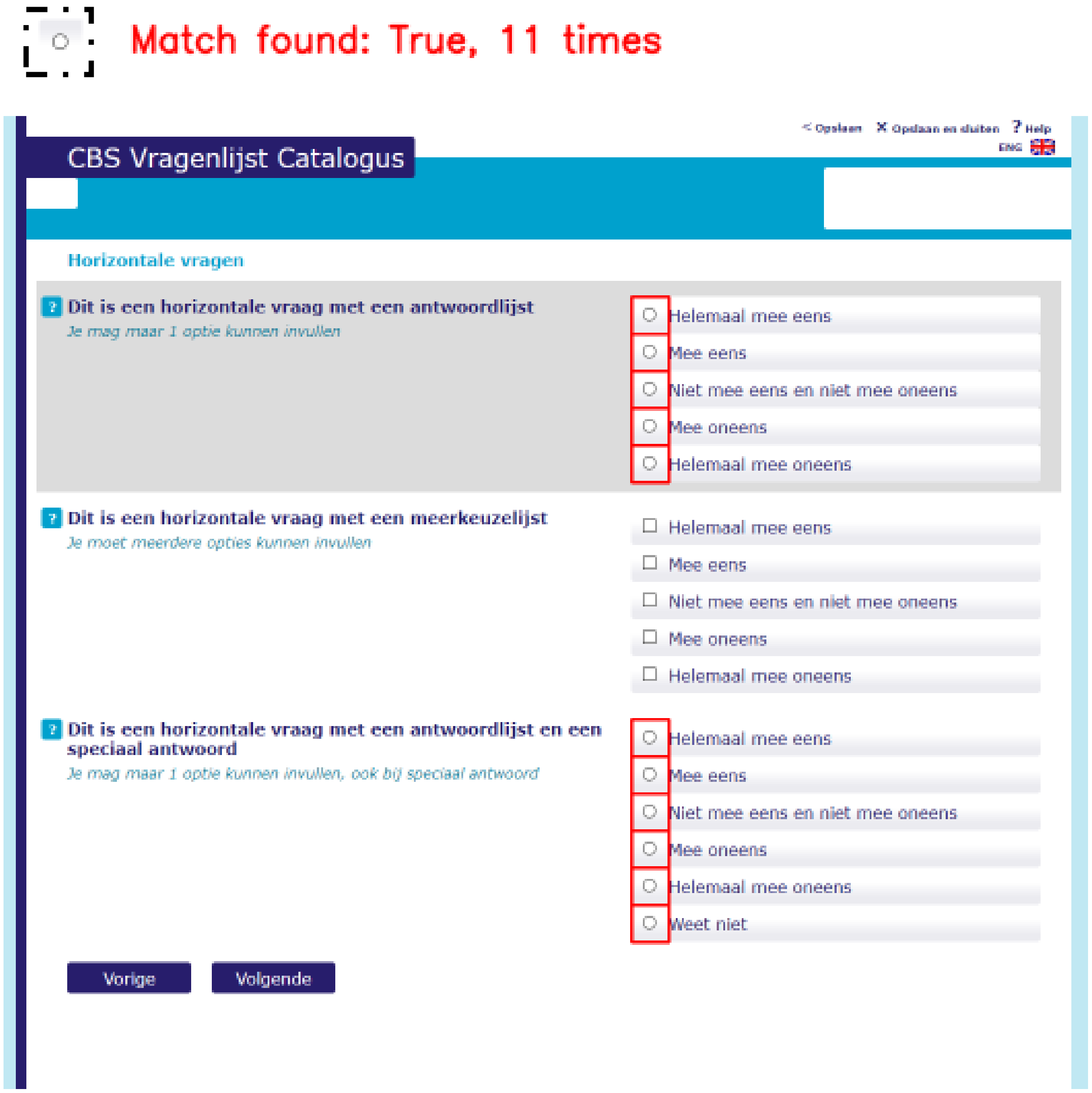


Figure 2 - template matching is a method for searching and finding the location of a template image in a larger image


Match found: True, 11 times

CBS Vragenlijst Catalogus

Horizontale vragen

Dit is een horizontale vraag met een antwoordlijst
Je mag maar 1 optie kunnen invullen

- ○ Helemaal mee eens
- ○ Mee eens
- ○ Niet mee eens en niet mee oneens
- ○ Mee oneens
- ○ Helemaal mee oneens

Dit is een horizontale vraag met een meerkeuzelijst
Je moet meerdere opties kunnen invullen

- ☐ Helemaal mee eens
- ☐ Mee eens
- ☐ Niet mee eens en niet mee oneens
- ☐ Mee oneens
- ☐ Helemaal mee oneens

Dit is een horizontale vraag met een antwoordlijst en een speciaal antwoord
Je mag maar 1 optie kunnen invullen, ook bij speciaal antwoord

- ○ Helemaal mee eens
- ○ Mee eens
- ○ Niet mee eens en niet mee oneens
- ○ Mee oneens
- ○ Helemaal mee oneens
- ○ Weet niet

4.5 Maintenance due to changing ID, XPath or other locators

The Blaisium framework uses multiple design patterns. The page object pattern is a decoupling layer that is used for a clean separation between general test code and page specific code such as locators. There is one repository for all services or operations offered by the specific page instead of spreading these services across all tests.

Instead of a test tool locating each test element directly and being vulnerable to UI changes, the Blaisium framework avoids this through the page object pattern. This makes it easier to understand and maintain the tests per test page.

5. Roadmap of Blaisium

5.1 Implemented

- Blaisium can build, install and uninstall a Blaise survey
- Blaisium web testing on local browsers Firefox and Chrome
- Blaisium web testing on multiple devices through BrowserStack
- Blaisium Windows dep testing
- Blaisium file compairing: .pdf, .txt, .xml, .png ...

5.2 For further development

- Blaisium app testing on Android through BrowserStack
- Blaisium app testing on iOS through BrowserStack
- Blaise API Calls