Bachelor Degree Project


Author: Erik Vasquez Petersson & Robert Forsgren
Semester: ST 2018
Subject: Computer Science
Abstract

Software testing is something that is very common and is done to increase the quality of and confidence in a software. In this report, an idea is proposed to create a software for GUI regression testing which uses image recognition to perform steps from test cases. The problem that exists with such a solution is that if a GUI has had changes made to it, then many test cases might break. For this reason, REGTEST was created which is a GUI regression testing tool that is able to handle one type of change that has been made to the GUI component, such as a change in color, shape, location or text. This type of solution is interesting because setting up tests with such a tool can be very fast and easy, but one previously big drawback of using image recognition for GUI testing is that it has not been able to handle changes well. It can be compared to tools that use IDs to perform a test where the actual visualization of a GUI component does not matter; It only matters that the ID stays the same; however, when using such tools, it either requires underlying knowledge of the GUI component naming conventions or the use of tools which automatically constructs XPath queries for the components. To verify that REGTEST can work as well as existing tools a comparison was made against two professional tools called Ranorex and Kantu. In those tests, REGTEST proved very successful and performed close to, or better than the other software.

Keywords: GUI, Test, Regression, Regression test, Machine vision, Google Cloud Vision, OpenCV, Adaptive, Automatic, Automation, Image recognition, Web Testing, Similarity.
Contents

1 Introduction 3
  1.1 Background 4
  1.2 Related work 6
  1.3 Problem formulation 8
  1.4 Motivation 9
  1.5 Objectives 9
  1.6 Scope/Limitation 10
  1.7 Target group 11
  1.8 Outline 11

2 Method 12
  2.1 REGTEST implementation. 12
  2.2 Controlled Experiments 14
  2.3 Reliability and Validity 16

3 Implementation 17
  3.1 Functional overview of how REGTEST works 17
  3.2 Frameworks, tools and libraries used in the implementation 18
     Java 8 18
     Gradle 19
     OpenCV 19
     Google Cloud Vision API 19
  3.3 Flowchart overview of test execution in REGTEST. 19
  3.4 GUI Buttons Overview 22
  3.5 REGTEST GUI overview. 23
  3.6 Test execution algorithm 24

4 Results 26
  4.1 Data Collection 26
     4.1.1 Tests run with no changes made to the system under test. 27
     4.1.2 Tests run with valid changes made to the system under test. 28
  4.1.3 Tests run with invalid changes made to the system under test. 28
4.1.4 Total success rate comparison.
1 Introduction

The introductory chapter gives a brief overview of the background of software testing and mainly GUI testing. The chapter goes into related research that has been done, and also mentions a few professional tools with some similarity, before, describing the problem to be solved as well as motivating why and to whom a solution to the problem is interesting. The primary objectives of this project are also listed and described.

1.1 Background

The focus in this degree project, from here on simply referred to as the project, is trying to solve an existing problem within the area of software testing. Software testing is something that is done to identify bugs in the form of defects, faults or failures in a software or a system [1]. It is important to identify these to improve the quality of the software and increase the confidence in the software. When it comes to software testing there exists a few different types, one such is scenario testing where a complete flow of actions is tested. One such scenario could be to login to a website, or to create and publish a message. Scenario testing is great for ensuring that not only single actions but complete workflows are working as expected. Another type of testing is regression testing which is to reuse previously created tests without modification when the software has been modified [1]. This is done to verify that defects have not been introduced into the software when it was modified and that the areas which were working before are still working as expected.

A Graphical User Interface, referred to as GUI, is something that could greatly benefit from implementing the above methods for testing. Any user-interface benefits from a test suite to ensure that it fulfills its use-cases. For a GUI that has a large number of possible paths through it, such a test suite can quickly become vast and time-consuming to develop and maintain since changes are continuously done throughout the development process. Even though a change may be small it can still break tests that, for example, depend on ID’s, or methods not changing, and this requires time and effort to look after and maintain. When new functionality is added to a software, it is essential to make sure that it is well tested both for what it is supposed to do
and also for what it is not supposed to do. However, if earlier tests are broken and need to be looked at, and rewritten because of the new functionality that was added to the software; Then it could take time and effort that was initially intended for creating tests for the new functionality. This could cause delays in the release of the software, or have an impact on the quality of the new functionality since it might not have had the time to be fully covered by tests. Therefore it is vital that care be taken when deciding on the nature of the test suite.

When creating a test suite for a GUI, there are several alternatives available to the test designer. The first choice is between a manual test suite and an automatic test suite. A manual test suite is one where test cases are documented and must then be manually executed by a human tester. Manual testing can be very time-consuming because as a software grows, so does the testing effort needed to maintain a high-quality software because the number of paths through a software increases exponentially. In an automatic test suite, however, the tester creates scripted tests that can be executed by a program, allowing them to be integrated as an automatic part of a workflow. This can save much time for the tester who then can focus on creating well tested and covered functionality, instead of performing manual testing which most likely takes a lot more time for a human than for a computer; Manual testing can also be very tedious for a tester to perform, since the same tests are repeated over and over again. If an automatic test suite is chosen, there are further choices which must be made. There are methods where the test cases are scripted to look at the underlying structure of the GUI, e.g., the DOM tree of a web application such as the Selenium framework [2]. Another method is image driven automation where the user specifies images and attaches actions to them to form a path that the program can follow to complete the test case. An example of an image-driven tool for automated GUI tests is Kantu [3].

The problem that this project aims to address is automating the process of detecting, what in this project is referred to as, valid changes that have been made to a software GUI. It should also allow for automated regression tests to notify the user of changes that have been detected before trying to continue running the test until it has either failed or finished.
1.2 Related work

Research has been done on areas close to the problem that this project aims to solve, and there exist many tools for automating GUI tests and also a few that use some sort of similarity functionality but not to the extent of this project.

Kantu is a tool for web testing, web automation, and web scraping [3]. It works similarly to the program outlined in this paper, using images and recorded actions as its driving force, instead of using the DOM and selectors to define elements and trigger actions. The key difference is that Kantu allows the user to set a similarity threshold, that can account for differences in rendering between devices. However, the program outlined in this paper, from now on referred to as REGTEST, will be able to handle a much higher degree of change and allow the user to adapt the test cases to said change instantly.

To mention a couple more tools for automating GUI testing, there is Selenium [2] which is a widely used tool that has been around for quite some time, and then there are newer tools like for example the previously mentioned Ranorex [4]. The difference between these two is that Selenium works on the DOM level, using IDs to navigate and perform actions, while Ranorex either uses image recognition functionality or both. Ranorex is a cross-platform tool for automating GUI tests, and it has a feature called Similarity Score which makes it more similar to this project than Selenium. Similarity Score works by allowing a user to set a similarity value which is a value between 1 and 0. The set value defines the range of similarity values that are acceptable [5]. Anything lower than the set value is discarded, and anything above the set value is considered to be a match and will allow the test to continue to be executed. The Similarity Score feature solves a problem that is similar to what this project aims to solve. However, as from what is written in the documentation for Ranorex, "The similarity value can be adjusted from 0.0 to 1.0. This corresponds to 0 % similarity (completely different pictures) and 100 % similarity (completely identical pictures)." [5] and "At 0.9 similarity, an entirely white 100-pixel picture would be considered identical to a picture with 90 white and 10 black pixels. That is quite a difference already. When you start comparing images in the magnitude of several thousand pixels, the optical deviations can be even more
striking." [5], it is clear that they are using some algorithm that only looks at pixel values and evaluates if enough pixels are of the same colors to pass the similarity value defined by a user. While this allows for small deviations, it does not allow for the same degree of change that REGTEST aims to handle, and at the same time automate, so that the user does not need to estimate and set such similarity values.

Something all these tools have in common is that they can help reduce the cost of testing web application GUIs. Still, research [6] has shown that there is a significant cost related to maintaining automated GUI tests, even if it can be reduced by doing frequent test maintenance instead of less-frequent but bigger maintenance.

Not much can be found that is entirely related to this project. Things such as tools for automating the creation of test cases using different methods, algorithms [7][8] and even with the help of AI [9], exist and have been researched previously but it is not similar enough to be of much help to this project. A big part of the project is, however, to understand the contents of 2D images and that is something that has been researched for many years now. In 1989 a Convolutional Neural Network, referred to as CNN, was designed and trained to recognize and classify handwritten digits [10]. A CNN is a multi-layered neural network which is designed especially for 2D image classification [11] and is commonly used to this day, or at least variations of it such as Deep Convolutional Neural Networks [12], also referred to as DCNN. Much research has been done since and new models which are variations of CNN's are being proposed.

Different types of neural networks have in common that they are used to classify images, but there are different levels of classification. An example of a classification task could be to identify different types of vehicles in an image. Another example is fine-grained object classification which is object classification on a lower level where classification is done on categories which are very similar. There is research that has been done in this area by using the same types of networks that very often are used to classify images, such as convolutional, deep convolutional neural networks or bilinear convolutional deep neural networks [13]. A typical case for when fine-grained classification could be used is to classify different types of plants, instead of giving results that every image is a plant [14].

Fine-grained classification is something that could be of great interest
when looking at the different types of web components which like many other things comes in many different types, shapes, and forms but which often are very similar looking.

An existing tool that can be used to understand the contents of an image is Google’s Cloud Vision project which is free to try [15]. It is very fast at analyzing an image and also has a web-mode where it can identify web entities in an image. Apart from it being able to identify web entities, it can also analyze and return information such as texts, colors, and classification labels which makes it the prime candidate to use for this project.

1.3 Problem formulation

The goal of this project is to investigate the possibilities of, and attempt to implement an adaptive GUI regression testing tool in combination with an artificial intelligence (AI) implementation. The reason for it is that no matter how automated a test suite is there might come a time when correct changes break an, as of that point, out-dated test case. If a user instead were to update and maintain all failing test cases manually, it can require much time, and possibly the need of involving more resources that have the knowledge of how specific tests are supposed to work. Another downside of manual testing is that a human requires rest while a computer does not. A human might also make errors and might not perform a test in the exact same way every time, which a computer, on the other hand, does since it has not been told to do it in any other way. The fact that a computer is both faster, and can perform the same task in the exact same way any number of times, but also requires no rest makes it very well suited for such tasks.

The focus of this project is to minimize the time spent adapting old test cases to new specifications by allowing the test tool to determine if a test breaking change actually is a valid change so that the test can continue. Currently, there exists no solution that can, to the same extent as this project, solve that problem. There does, however, as mentioned in related research, exist tools that have functionality that can solve simpler cases of change.

As discussed in related work, even though a project has a fully automated GUI test suite using currently available tools, there is still a significant cost associated with maintaining such a test suite. We believe that
REGTEST could help minimize maintenance costs by allowing developers to quickly review and approve the changes detected by REGTEST, thereby integrating test case maintenance as a part of the verification process.

1.4 Motivation

There exist many tools for automating GUI testing and these tools have in common that they are used to save time, improve maintainability and the quality of the software. However, there exists a problem which is that when changes have been made it can very easily break test cases, which then has to be updated and that takes time and effort from the testers.

Automation tools for creating test cases with the help of AI has been researched. However, we have not found any automated GUI testing tool that has integrated an AI together with image recognition, or any other solution allowing for the same degree of adaptability when it comes to changes, and without breaking test cases.

Many GUI test automation frameworks rely on a DOM or similar to function, using element classes and IDs to navigate. One of the most significant drawbacks of this method is its reliance on strict naming conventions for all the GUI components. In any project, there is always a risk of the decided upon naming conventions not being followed [16] all the time. When using a test framework that depends on strict naming conventions to function, failure to follow said conventions easily add more operating costs for the project when old test cases have to be updated, or new additions to the codebase be reworked for failing to follow conventions.

1.5 Objectives

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>O1</strong></td>
<td>Select the best-suited type and method for creating and implementing the AI solution.</td>
</tr>
<tr>
<td><strong>O2</strong></td>
<td>Implementation of the AI solution.</td>
</tr>
<tr>
<td><strong>O3</strong></td>
<td>Implement a basic testing tool which allows the user to set up a flow of actions to be performed on a webpage, and incorporate the AI.</td>
</tr>
</tbody>
</table>
Collect data on how successful REGTEST is in detecting valid changes; Compare the data to the results of Ranorex and Kantu.

Table 1.1: Objectives.

There are four primary objectives set for this project (Table 1.1) and the expectations for Objective 1 is to find the best-suited type and method for implementing the AI, which must be possible to implement within the scope of the project.

The expectation for Objective 2 is to have a complete AI solution implemented. In combination with other tools and algorithms the AI should be able to look at an image that has changed, as well as its original, and then decide whether it is the same web component in both images even if a change may have occurred.

The third objective is to find an existing tool or framework which can be used to perform actions such as moving the cursor and clicking on a specified location. The AI will then be incorporated in the implementation of the action tool or framework to allow for a user to set up a flow of actions to be performed as a test. The AI solution will be implemented as one such action, and depending on the result returned from the AI the testing tool will either continue performing its defined set of actions, or stop. If all actions are performed as expected, then the test will be considered to have passed; otherwise, it will not perform any more actions, and the test will be considered to have failed.

The final objective is to test and compare how well REGTEST can detect valid changes. The same tests will then be run in and compared to the results of Ranorex and Kantu, with their respective similarity functionality to determine if REGTEST, with the help of AI, can perform better than the similarity functionality that currently exists on the market.

1.6 Scope/Limitation

Within the scope of this project, the resulting program should allow the user to setup test cases consisting of recorded inputs and images acting as checkpoints. Using the programs ability to discern whether a change is within
the accepted threshold, which will be limited to one change at a time, out of four possible types, shape, color, location or text, it should be possible to move the test baseline forward if the GUI being tested has changed according to new specifications, but still stayed within the change-threshold.

Since the project is limited in time, some of the features present in similar, commercially available tools will be left out. Examples of this will be the lack of integration with Jenkins [17] and other such continuous integration software and the fact that REGTEST will be limited to testing web applications. Furthermore, in case REGTEST shows different behavior depending on which web browser it is run in conjunction with, Google Chrome will be the web browser primarily supported.

1.7 Target group

Maintaining high test coverage of functionality is a good measure to prevent software regression. However, achieving such coverage comes with an upfront cost of development time, and maintaining it brings further costs. For very time-sensitive projects this cost might lead developers to skip implementing them altogether. Therefore, REGTEST outlined in this project could help development teams of any size that work with web applications, to quickly set up regression tests at a low cost, and to maintain them with little effort.

1.8 Outline

The next chapter describes a technical overview of the implementation, before explaining the controlled experiments that will be performed and the reliability and validity of those results. Following is the implementation chapter where more technical insights, as well as an overview of how REGTEST works, are presented. The remaining chapter analyses and discusses the results before writing out the conclusions made and possible future work that could improve this, or similar solutions.
2 Method

This chapter describes how REGTEST will be implemented, how its success in addressing the problem formulation will be measured and any potential reliability and validity concerns.

2.1 REGTEST implementation.

REGTEST will be developed to be able to perform common actions (e.g., Clicking, Typing) on a web page, and allow for a user to set up a flow of actions (Figure 1.1) to be performed for each test.

The choice of an AI that could be implemented within the scope of the project fell on the Google Cloud Vision (GCV) API. The GCV API will be used to get information about images taken on a web component when a test is set up. In conjunction with GCV, the open-source computer vision framework OpenCV will be used for its template match capabilities, which allows it to search for an image within another image, and return its position. With all the information that it is possible to get from the GCV API together with the information from the template match, an algorithm will be created to try and determine if a valid change has occurred. The algorithm will work by comparing the GCV results and template matcher results of the image that was taken when the test was created, and the results of an image taken upon running the test. If the algorithm detects no invalid changes, then REGTEST will continue with the test.

A change is considered invalid if it contains more than one type of change, with the different types of changes being: a change to the text contained in a component, a change of color in any part of the component, the component changing location and a change in the shape of a component (Figure 2.2). The restriction of one change at a time is to limit complexity and fit REGTEST inside the scope of this project, since allowing multiple changes of a component at the same time has the potential of making it much harder to identify the original component, e.g., a menu item changing both its text and location.

Since Google Cloud Vision is already trained, owned, and controlled by Google, there is no need to create test data for the purpose of training the AI. Instead, live web pages will be used to verify that REGTEST works
before using a new set of test data for the comparison of all three tools, referred to in chapter 1 as the fourth objective.

Figure 2.1 Example of a user-specified flow of actions which are to be performed by REGTEST.

Figure 2.2. Example of valid changes.
2.2 Controlled Experiments

For this project, a series of controlled experiments will be performed to measure how well REGTEST can handle different type of changes. There will mainly be three different experiments which are:

1. Tests with valid changes made to web components, where only a single web component will be tested in each test.
2. Tests with invalid changes made to web components, where only a single web component will be tested in each test.
3. Tests with complete workflows where no changes have been made, but where multiple web components will be tested in each test.

Neither Kantu nor Ranorex is using AI in their similarity features; However, it can still be of great interest to perform a comparison against both of them. Therefore all three experiments tested on REGTEST will also be tested on both Kantu and Ranorex resulting in a total of 9 experiments.

The experiments will consist of test cases built up with different sets of image pairs for each type of change that REGTEST will be tested on. The test cases will consist of pairs of images which REGTEST will then evaluate, and determine if the image pair are both images of the same component but which has a valid change made to it.

As mentioned, each experiment will have their own set of image pairs that will be focused on one type of change. However, each set of images will have images of web components of different shapes, sizes, colors, positions and with different texts. This is to test with many variations of common web components and get more accurate results.

An example of the different variations of web components that could be tested, where the shape has changed, is described in the table (2.1) below. The table describes a valid change as well as how the images may vary between different image pairs without being considered an invalid change.
Valid Shape Change: The shape must be different between the original image and the new image to be considered a valid change, but the color, location or text of the web component may not be different between the original image and the new image.

Image Variation 1: A web component may be of any color, as long as it is of the same color for both the original and the new image. It would otherwise be considered invalid since both the shape and color would have changed.

Image Variation 2: In the same way, as with the colors, a text in the images is allowed to be different for different image pairs, but not between the original and new image.

Image Variation 3: The location of a web component may be anywhere within the visible area of a web browser, but may not have changed location between the original and the new image. It would then be considered invalid since shape and position has changed.

Table 2.1. Description of the type of variations that are allowed and can be used in the experiments containing web components of different shapes.

Quantitative data will be collected and measured from the results of REGTEST to determine how well it can handle the type of changes. What is expected after each experiment is statistical data on the following:

- How many cases of valid changes were correctly identified and handled by REGTEST?
- How many cases of valid changes were not identified and handled by REGTEST?
- How many cases of False-Positives did REGTEST report?

This data can be compared against the other experiments to determine the easiest, and the hardest changes that REGTEST can or cannot handle, as well as how well REGTEST perform compared to the other tools. It can unfortunately not be compared against results from other tools that use AI to determine whether it is an accepted change or not since no other similar tool could be found.
2.3 Reliability and Validity

The work in this project can be considered to be reliable for the reasons that all code and algorithms used in the project are open-source, meaning that the exact code and algorithms can be pulled down from the Git repository and run to replicate and verify the results. However, it requires that the same test cases are run, and in the same environment set up, meaning the REGTEST is run on the same operating system using the same browser, then the results should be replicable.

A potential issue is with the AI being used; Depending on the AI solution chosen, there might be no control over further training of the AI. In the future, this could lead to behavioral differences in the REGTEST when handling changes within the allowed threshold, and the results outlined in this paper no longer being exactly replicable.

For the validity of the comparison results of REGTEST against Ranorex and Kantu, it must be understood that neither Ranorex nor Kantu claims to be able to handle the same type of changes that REGTEST do, nor do they work in the same way by using AI to determine the similarity of a web component in two images. However, they both have features that are closely related to what REGTEST does, and they are the tools with the most similar feature, which therefore makes them interesting to compare against.

To simulate real-world use-cases and ensure a varied data-set, the images used for the controlled experiments will be from a visually diverse set of popular websites with complex GUIs. However, this comes with a drawback. Due to time constraints, the data set used in the experiments will have to be smaller than if images of simple web elements were used, something which could have been generated in the thousands with little effort.
3 Implementation

This chapter will provide an overview of how REGTEST works, which third-party technologies that were used, what the REGTEST GUI looks like and provide a closer look at how the test execution algorithm was implemented.

3.1 Functional overview of how REGTEST works

REGTEST is an automated GUI regression testing tool, and it works by allowing a user to set up one or more Test Groups, which is a collection that can contain several tests. Tests can then be created in a new, or a previously created Test Group. During the creation of a test, the user has to type in the URL to the web page that is to be tested which will be considered as a first step of the Test that REGTEST will open when running the test.

Once a test has been created, the user has the option to either start what could be called a test-recording. If a user presses the Record button, then REGTEST will open a Google Chrome browser on the primary monitor and then navigate to the user-specified URL. Once the page has opened and loaded, the user gets to choose between a number of actions that will be performed as the next step of the test. Once an action has been selected and set up if needed, the user can then continue to add actions to create a flow of steps that will be performed by REGTEST when the user runs the test. The record button changes to a stop button during the recording, which is then used to both stop the recording, meaning that no more steps can be added to the test, and also saves the test.

To be able to set up tests similar to how a real user would interact and use a web page we have implemented five main actions that can be added to build up the different steps to be performed to fulfill a full test.

1. The first action is the Browser Opener that automatically gets added when a new test is created. The Browser Opener opens and navigates to the specified URL.
2. The second action is the Click Action which allows the user to specify a point on the primary monitor that REGTEST should click on, before moving on to the next step.
3. The third action is called the Snap Image Action, and if added to a test it allows a user to click, hold and drag a rectangle on the primary monitor, and when released REGTEST will take an image of that area and store it to the test. When such a test step is run, REGTEST will then click on the image.

4. The fourth action is the Type Action which lets the user enter a text that REGTEST will enter when running the test, and specific test-step, is run.

5. The fifth and final action is a timer that can be used to add a user-specified wait time between test-steps.

When one, or several tests have been created, they can either be run one by one or as an entire test-group could be selected and then all the tests contained in that test group will be run. When a test is run, REGTEST will open the browser and go through all the test-steps with its user-defined actions before displaying the test results to the user of all the tests that have been run. The results are presented in such a way that if everything worked out as expected, the user simply gets notified that the test was successful. If on the other hand, a test contained what the software considers to be a valid change then the user will be presented with what has changed, and also be given the option to update the old baseline to use the new image for the next time that the test is run. If a test completely fails, then the user will get notified of the specific test-step that failed.

Additional information worth mentioning is that REGTEST saves all tests after a recording is finished. When REGTEST saves a test, it creates a folder containing a .json file with all the information about a test, together with a sub-folder with the images used for the test.

3.2 Frameworks, tools and libraries used in the implementation

There are a number of different frameworks and tools that were used in the implementation of REGTEST, and the ones that are worth mentioning are all listed below.

**Java 8**

REGTEST is a project written in Java using the Java Development Kit 8, and
the GUI was created using JavaFX 8.

**Gradle**
Gradle, an open-source build automation system was selected to download and keep dependencies updated as well as to build the software.

**OpenCV**
We use the open-source library OpenCV for its computer vision capabilities, to be more exact, the template matching feature. Template matching takes two images, and an optional mask as input and returns the most likely position of the second image inside the first image. Since the images are always square-shaped, a mask can be used to find custom shapes from the second image.

**Google Cloud Vision API**
The choice of selecting the most appropriate AI solution for the project fell on the Google Cloud Vision (GCV) API. This is for two main reasons, which are to be able to implement it within the scope of the project, but also because the GCV API makes it easy to get different types of data from an image, such as text, colors, and properties referred to as labels. It also has the capability to detect web entities and similar to the labels detection, it returns properties, but from an AI trained specifically for web entities. The combination of information that is possible to get from the GCV API makes it a very good choice that will fit within the scope of the project.

### 3.3 Flowchart overview of test execution in REGTEST.

Figure 3.1 displays a flowchart overview of a full test execution focused on test-steps using the Snap Image Actions. However, a brief explanation of the terminology used might be necessary and can be seen in Table 3.2.

<table>
<thead>
<tr>
<th><strong>Common REGTEST terms</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-group</td>
<td>A group or collection of tests.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Test</td>
<td>A test contains a number of user-specified steps that REGTEST runs on a web page to verify a workflow.</td>
</tr>
<tr>
<td>Test-step</td>
<td>A Test-step is added to a test and contains one main action. All test-steps in a test are performed in synchronous order.</td>
</tr>
<tr>
<td>Target / Target Image</td>
<td>A Target or Target image in REGTEST is the image that a user has taken which REGTEST tried to find and click on when running a test.</td>
</tr>
<tr>
<td>Context / Context Image</td>
<td>Context images are images of the entire visible webpage. All steps contain a context image.</td>
</tr>
</tbody>
</table>

Table 3.2: Common Terms.
Figure 3.1: Flowchart view of test execution in REGTEST.
### 3.4 GUI Buttons Overview

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Browser Opener Action icon" /></td>
<td>Browser Opener Action</td>
<td>The Browser Action gets added to a test by default. When run, it uses the user-specified URL that was set when the test was created.</td>
</tr>
<tr>
<td><img src="image" alt="Click Action icon" /></td>
<td>Click Action</td>
<td>The Click Action allows the user to specify a screen coordinate on the Primary Screen which REGTEST will click on when run.</td>
</tr>
<tr>
<td><img src="image" alt="Snap Image Action icon" /></td>
<td>Snap Image Action</td>
<td>The Snap Image Action allows the user to snap an image and save it as a target image for the selected test. The user can also set where, within the image, that the click should be performed. When run, REGTEST will attempt to find the target image in the browser and click on it.</td>
</tr>
<tr>
<td><img src="image" alt="Type Action icon" /></td>
<td>Type Action</td>
<td>Adds a Typing Action to the selected test which will type a user-defined text when run.</td>
</tr>
<tr>
<td><img src="image" alt="Timer Action icon" /></td>
<td>Timer Action</td>
<td>Adds a timer to the selected test, which will be performed before continuing to the next test-step.</td>
</tr>
<tr>
<td><img src="image" alt="Create New Test icon" /></td>
<td>Create New Test</td>
<td>Displays a dialog which allows the user to create a new test, and either select a test-group which the test should be saved to, or create a new test-group.</td>
</tr>
<tr>
<td><img src="image" alt="Start Test Recording icon" /></td>
<td>Start Test Recording</td>
<td>Starts a recording for a selected test, allowing for actions to be added.</td>
</tr>
<tr>
<td>![Stop Test Recording]</td>
<td>Stop Test Recording</td>
<td>Stop Recording - stops the ability to add actions to a test. Saves the test in its current state.</td>
</tr>
<tr>
<td>![Run Selected Test(s)]</td>
<td>Run Selected Test(s)</td>
<td>Runs the selected test, or if a test-group is selected, it will run all tests belonging to that test-group.</td>
</tr>
<tr>
<td>![Preferences]</td>
<td>Preferences</td>
<td>Preferences are mainly used to set the path to where tests should be stored, as well as to store the path to the Selenium Chrome Driver needed to run a test in REGTEST. The preferences window also contains the threshold values used by the GCV algorithm.</td>
</tr>
</tbody>
</table>

Table 3.2: Overview of the most commonly used buttons in REGTEST’s main window.

3.5 REGTEST GUI overview.

Figure 3.2 displays an overview of REGTEST when having a test selected from the test-group GOOGLE. The second test-step is selected, from the narrow vertical list of icons which is of the type IMAGESNAP. When selected the IMAGESNAP step displays the target image and the context it was taken in, as well as information for both images.
Figure 3.2: REGTEST's main window with a test selected, and one of its test steps, which is of the type IMAGESNAP.

3.6 Test execution algorithm

Tests in REGTEST are saved as a list of test steps to be executed sequentially. Each step contains an action, and the action contains a context image and a target image. When a target image is taken and stored, a call to the GCV API is also made to collect and store data for the target image. At present, there are currently six types of information that are collected and stored from the GCV API together with a score value which determines how certain the GCV is of the returned results.

- Labels
- Web Entities
- Dominant Colors
- Text
- Text Crop Hints
- Logos
When REGTEST executes the action of a test step the first thing it will do is to create a new image from the bounds of the original target image before sending it to the GCV API to collect information about the new target. All the information gathered is used to determine if the images are considered to be the same or if a change between the two images could be classified as a valid change.

The algorithm for verifying if a change is valid or invalid uses the score saved with some of the results collected from the GCV API to sort out low score results. The remainder is then compared and matched between the old target image and the new target image. If enough matches, which is determined by a user-specified value, are found for a resultset from the GCV API then it will be considered to be either a valid change or the same image. If, on the other hand, there are not enough matches then it would be considered to be an invalid change. If enough changes are present, then the GCV algorithm will consider the two target images to be too different and therefore determine that the component has been changed too much to be confident that it is the same component in both images.

If the GCV algorithm fails to find a match on the saved target image's location, REGTEST will use the matchTemplate() function of OpenCV to look for the target image in other parts of the context image. There are several matching methods available for use with matchTemplate(). REGTEST uses all available methods to ensure the highest possible chance of finding a match. However, the methods may sometimes produce different results compared to each other and because of this REGTEST will sort them and only use the results that are produced by the most matching methods. If the number of matching methods that give the same result is below a certain threshold, REGTEST will consider it as an unreliable result and report that no match was found for the target image.

If both the GCV algorithm and OpenCV template matching fail to find the target image, REGTEST will stop current test execution and save information regarding the test failure for later review by the user, as soon as any remaining tests have finished executing.
4 Results

This chapter explains the type of data that was collected, how it was collected as well as how it was measured. The data is displayed in diagrams and tables to give a clear overview of the results.

4.1 Data Collection

The data collected and presented in this chapter was collected to compare the three previously mentioned tools, which is REGTEST, Ranorex, and Kantu. A test suite with identical tests was created in all three programs, and this is an entirely new set of data that was created for the software evaluation only. The test suites contain three different types of tests which are:

1. Tests that are expected to pass with no changes made to the web components.
2. Tests that are expected to pass with valid changes made to web components.
3. Tests that are not expected to pass with invalid changes made to web components.

The test suites were created in all three software to compare how well REGTEST stands up to Ranorex and Kantu which are two professional tools. It is also done to compare if the solution of having an AI, together with Template Matching, can perform as well, or better than the two professional tools of Ranorex and Kantu.

The collected data is measured in Test Steps, and the number of Test Steps that the three different software was able to perform successfully, out of the total number of Test Steps. The different types of web components contained in the test suite are:

- Text / Hyperlinks
- Buttons
- Checkboxes
- Images
- Dropdown Menus
Table 4.1 to 4.4 shows data for all three tools and how many steps that they were able to perform successfully. It also displays the total number of steps together with the success rate in percentage. Figure 4.1 to 4.4 all display the number of successfully performed Test Steps together with the Total number of Test Steps.

4.1.1 Tests run with no changes made to the system under test.

<table>
<thead>
<tr>
<th></th>
<th>Steps with No Change</th>
<th>TOTAL</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGTEST</td>
<td>40</td>
<td>41</td>
<td>97.56</td>
</tr>
<tr>
<td>RANOREX</td>
<td>40</td>
<td>41</td>
<td>97.56</td>
</tr>
<tr>
<td>KANTU</td>
<td>41</td>
<td>41</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Table 4.1: Test steps with no changes.

Figure 4.1: Number of Test Steps, with no changes made, that was successfully performed.
4.1.2 Tests run with valid changes made to the system under test.

<table>
<thead>
<tr>
<th></th>
<th>Steps with Valid Changes</th>
<th>TOTAL</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGTEST</td>
<td>41</td>
<td>44</td>
<td>93.18</td>
</tr>
<tr>
<td>RANOREX</td>
<td>15</td>
<td>44</td>
<td>34.09</td>
</tr>
<tr>
<td>KANTU</td>
<td>23</td>
<td>44</td>
<td>52.27</td>
</tr>
</tbody>
</table>

Table 4.2: Test Steps with valid changes.

Figure 4.2: Number of Test Steps, with valid changes made, that was successfully performed.

4.1.3 Tests run with invalid changes made to the system under test.

Eight tests, with a small number of changes, were used only for the purpose to give an idea of how more than one change, which is considered an invalid change, would be handled by the three tools. The matches are seen as invalid, even though the correct web component may have been found.
### 4.1.4 Total success rate comparison.

<table>
<thead>
<tr>
<th></th>
<th>Successful steps</th>
<th>Total Steps</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGTEST</td>
<td>81</td>
<td>85</td>
<td>95.29</td>
</tr>
<tr>
<td>RANOREX</td>
<td>55</td>
<td>85</td>
<td>64.71</td>
</tr>
<tr>
<td>KANTU</td>
<td>64</td>
<td>85</td>
<td>75.29</td>
</tr>
</tbody>
</table>

Table 4.4: The total number of successfully performed Tests Steps and the total amount of Test Steps. Steps with invalid changes excluded.
Figure 4.4: The total number of successfully performed Test Steps.
5 Analysis

The goal of REGTEST is mainly to be able to adapt to a single change in web components, but it also has to be able to handle regular workflows where there have been no changes made to any web components. The most interesting results gathered is how well the three different tools were able to complete a regular workflow, as well as how well they could adapt and handle cases where one change had been made to a web component.

Looking first at the result for when no changes have been made to the components (table 4.1 & figure 4.1), then Kantu was the software that performed the best and was able to perform 100% of the test steps successfully. However, both Ranorex and REGTEST come close in second place with 40 successfully performed steps out of the total of 41 steps, with a 97.56% success rate. For the case when no changes have been made it is expected that a GUI regression testing tool should successfully be able to perform all the steps without any failures. The problem that was encountered with Ranorex was that after creating a complete test with their image-recognition tool, for most cases, the user had to manually adjust the parameters to be able to get a match when running the test, even though no changes had been made. For this reason, it is possible that the limited experience that we have working with Ranorex is the cause of Ranorex not finding this specific component. We did, however, spend a good amount of time attempting to adjust the parameters but to no success, instead, we either found no match at all or an incorrect match in the wrong position. The problem that exists with REGTEST is believed to be due to slow loading times of the locally stored web pages that were used for the test. This is caused by long execution time of included Javascript, and that because of it, an image was taken and compared against the old target before the page had been fully loaded. Since REGTEST is somewhat a proof of concept, things such as this which could be considered as bugs are bound to show themselves and, even though REGTEST has a nearly perfect match rate for the test suite used, this indicates that more testing needs to be done.

The second result, which for REGTEST is the most interesting to look at, is how well Valid Changes could be handled by all three tools (table 4.2 & figure 4.2). Here REGTEST comes in a clear first place with its 41 matches out of 44, with a success rate of 93.18%. Kantu comes in second place with
23 matches and a success rate of 52.27%, and Ranorex in third place with 15 matches and a success rate of 34.09%. As mentioned earlier, neither Ranorex nor Kantu are tools that claim to be able to handle all these different types of changes, but mainly just when the image has got a new position on the page. Ranorex does, however, have filters and settings that can be used when setting up the test to allow for some adaptability, though, it is also often that those settings are adjusted just to get the test to successfully be performed without changes. Since a future change is not always known in advance, adjusting the parameters to try and predict the type of changes that may occur will most likely be a lot more time-consuming than just re-recording the step that failed.

The result of the tests containing invalid changes made to web components (table 4.3 & figure 4.3) is, despite the limited test data, something that we would like to mention. Eight tests were created in all three tools with components that had two or more changes made to them, and REGTEST was the software that found the most matches, 4 out of 8. These matches, although they are correct in the sense that the correct component was found, they are, for this project, considered as invalid since there have been too many changes made to be confident in the matching result. Kantu comes in a close second place with 3 out of 8 matches found, and Ranorex as third with 1 match out of the 8 in total. These results could indicate that some more adjusting and tweaking need to be done with the algorithm in REGTEST to only allow for a test step to pass if it is very confident in the match.

The test data collected showed both expected and surprising results. As expected, both Kantu and Ranorex proved very capable of reliably executing a recorded test once it had been set up, and in some cases, tweaked a little. What was surprising with the results, was the ability of Kantu and Ranorex to find matches for recorded images when the original elements captured in the images had been changed in various ways. In some cases even when it had changed in multiple ways at once; On the other hand, there were cases when valid changes to the original element caused them to instead pick a nearby element that looked similar to the original element, and in the worst cases not very similar at all. Since they lack the built-in safeguards against this that REGTEST has, it is possible for them to execute erroneous test flows without the user getting notified of this.
6 Discussion

When comparing REGTEST to the commercially available GUI testing tools Kantu and Ranorex, it is clear that REGTEST is of similar capability in regards to executing GUI test cases, getting a score of 97.56% (Table 4.1) when running against our test scenarios. Considering the limited time available for implementing REGTEST in this project, if there had been more time, further improvements to the REGTEST algorithm could very likely have improved this further. Where it gets interesting is when the results of test executions after valid changes are compared. After reviewing the sales material for Kantu and Ranorex, we never got the impression that they would be able to handle changes to the extent that they did, with at least Kantu only claiming the capacity to handle the rendering differences between different browsers [3]. One possible explanation as to why this was never mentioned could be the lack of feedback provided when they detected an image that had changed with regards to the original set up of the test, whether it is by a limitation of implementation or intentional. Without this kind of feedback, small design bugs could appear without the tester ever being made aware. This issue is one of the key features that separates them from REGTEST while also being one of the core problems this project aimed to address. With REGTEST the tester is made aware anytime REGTEST detects that the image recorded in a test step has changed in any way, thereby always ensuring that the tester makes a conscious decision to approve the change as correct or is otherwise able to correct the change. While data extracted from the live usage of REGTEST would be needed to prove it, it could be estimated that this ability to continuously, and with little effort, keep tests up to date with changing specifications will help bring down test suite maintenance costs [6].
7 Conclusion

In this project, an idea of creating a GUI regression testing tool with image recognition was proposed. The idea with the tool was that it had to be able to handle, what is referred to as, valid changes made to web components such as changes in text, color, or size. From this idea REGTEST was created, a GUI regression testing tool using a combination of the Google Cloud Vision API and OpenCV methods to gather as much relevant information as possible for REGTEST to determine if two images are actually of the same component and that a valid change has been detected.

REGTEST was compared to two other professional GUI regression testing tools and the results from that show that REGTEST performed as good or close to as good on regular tests where no changes had happened. However, when comparing the tools on how well they could handle valid changes then REGTEST performed the best and proved that this way of approaching this type of a problem is viable to some extent. However, the algorithm for recognizing valid changes versus invalid changes may need more work as indicated by the results of the tests with invalid changes made to web components. The benefit when using REGTEST instead of Ranorex or Kantu when any changes have been made, valid or invalid, is that REGTEST lets the user know what type of change it has detected, allowing for the user to decide if it is correct or not. There were several cases with both Ranorex and Kantu where no information was presented about changes that were detected.

Even though this report focused on testing web pages, the solution proposed could work for different types of platforms, such as web pages, desktop application, and mobile applications. The idea for REGTEST is simplicity and time saving, allowing for a user to create a test case by simply performing the workflow through REGTEST once, and without any knowledge of how the code works or what IDs different GUI elements has. Maintenance is kept to a minimum by allowing users to update test cases with the click of a button when REGTEST detects valid changes.

To show that REGTEST could measure up against other tools with similar functionality objective 4 was created "Collect data on how successful REGTEST is in detecting valid changes; Compare the data to the results of
Ranorex and Kantu.". Because of objective 4, the test data used for comparing the three tools had to be made smaller so that setting up all the tests, in all three tools, and gathering the results could be done within the scope of this project. Because the results are based on a relatively small set of test data, there is a need for more testing to be done, but REGTEST could be seen as a successfully implemented proof of concept.

There are a few things that could have been improved in REGTEST if time would have allowed, such as the fact that Google Chrome is the only web browser that it can work with. However, similar to Kantu, it does not handle long loading times perfectly which is thought to be why one test step failed for a regular workflow where no changes had been made to the component.

7.1 Future work

Because of the limited scope of the project, choices were made to use methods and tools that could be implemented within the scope. One of those choices were to use the Google Cloud Vision API to gather information about an image. For future research and testing, creating and training new neural networks, more focused on GUI components could be very beneficial to avoid getting information that is irrelevant for this type of a problem. Other benefits of creating neural networks are that control can be taken over what type of data the networks are trained with, and continuous learning [18] could be applied to adapt to potential trends and changes that might be common to GUI components in the future. A drawback of using GCV is that it is trained with images of various other things than just web components. Because of this, GCV could potentially send back information of what it thinks is a blue, rectangular speaker, when in fact it is actually a blue and rectangular web component. The information that is received from GCV is a bit to general to be able to use it as much as we would have liked. However, if neural networks were to be created for the purpose of only being able to detect different types of web components, colors, shapes and such information that for this project is of great interest, then the algorithm could potentially be improved considerably. The confidence in the results of neural networks could be increased by a lot since if a neural network only has been trained on
data that is relevant the project, then it would not be able to give information about a blue speaker since it has not been trained to detect it.

To increase the reliability of this type of a solution, there is a need to create more test data and identify more types of components as well as more types of changes that could occur to test as many different variations of valid changes as possible. Research could also be done on the types of change that are most commonly made to GUI components for web pages, but also desktop applications and mobile applications if multi-platform solutions are to be created.
References


