Investigating GUI test automation ROI: An industrial case study

Mälardalens Högskola
School of Innovation, Design and Technology
Rebecka Laurén
Bachelor thesis - DVA331
10-06-2015
IDT Examiner: Adnan Causevic
IDT Supervisor: Sara Abbaspour
Abstract
This report is the proof of concept that shows how Coded UI can be used for the automation of GUI tests. For this thesis-work, an industrial case study is done on 26 different test cases provided at the company Basalt AB. The problem they had was that testing was very time consuming and was therefore not done as often as needed to keep the level of quality required for the system developed. The method that has been used are called the validation method and the work was according to that divided into five steps: choosing test cases, a learning process, implementation, compare the results and then draw conclusions. Test automation has benefits of time savings and the fact that tests will be reused. Testing will take minutes instead of hours and the tests can be executed over and over again as many times as needed. So by changing from manual testing to automated testing, tests can be run faster and therefore more often. The investment of automated testing will be repaid before the end of the fourth test round. So conclusions can be drawn from the results of this thesis and it shows that it absolutely is worth the investment of automating the testing process.
Contents

1 Introduction ........................................................................................................................................ 5
  1.1 Concept explanation .................................................................................................................. 5
    1.1.1 Software testing .................................................................................................................. 5
    1.1.2 Regression testing .............................................................................................................. 5
    1.1.3 GUI testing .......................................................................................................................... 5
    1.1.4 ROI ...................................................................................................................................... 5
    1.1.5 Test Case ............................................................................................................................ 5
  1.2 The company and SWECSS ........................................................................................................ 6
  1.3 Tools ........................................................................................................................................... 6
    1.3.1 Microsoft Office SharePoint Server (MOSS) ....................................................................... 6
    1.3.2 Coded UI ............................................................................................................................ 6
      1.3.2.1 Coded UI test builder .................................................................................................. 6
      1.3.2.2 UI Maps ...................................................................................................................... 7
      1.3.2.3 Assertions ................................................................................................................... 7
  1.4 Problem definition ..................................................................................................................... 8
  1.5 The goal ..................................................................................................................................... 8

2 Background ..................................................................................................................................... 9
  2.1 State of the art ............................................................................................................................ 9

3 Methodology .................................................................................................................................. 10
  3.1 Initial time plan .......................................................................................................................... 11

4 Implementation ............................................................................................................................. 11
  4.1 Test case 1 ................................................................................................................................ 15
    4.1.1 Automation process .......................................................................................................... 15
    4.1.2 Lessons learned .............................................................................................................. 15
  4.2 Test case 2 ................................................................................................................................ 16
    4.2.1 Automation process .......................................................................................................... 16
    4.2.2 Lessons learned .............................................................................................................. 16
  4.3 Test case 3 ................................................................................................................................ 16
    4.3.1 Automation process .......................................................................................................... 17
    4.3.2 Lessons learned .............................................................................................................. 17
  4.4 Test case 4 ................................................................................................................................ 18
    4.4.1 Automation process .......................................................................................................... 18
  4.5 Test case 5 ................................................................................................................................ 19
    4.5.1 Automation process .......................................................................................................... 20
    4.5.2 Lessons learned .............................................................................................................. 20

5 Results .......................................................................................................................................... 20
  5.1 ROI .......................................................................................................................................... 21
  5.2 Site constructor ......................................................................................................................... 22
  5.3 Military time code ...................................................................................................................... 23
  5.4 Time .......................................................................................................................................... 23
  5.5 Copy list item ............................................................................................................................ 24
  5.6 Limitations ................................................................................................................................ 25

6 Discussion ..................................................................................................................................... 25
  6.1 Maintenance and guideline ........................................................................................................ 29

7 Conclusions ................................................................................................................................... 29
  7.1 Future work ............................................................................................................................... 30

References ......................................................................................................................................... 31
Table of figures
Figure 1: Coded UI Test Builder ........................................................................................................6
Figure 2: UI Map structure ..................................................................................................................7
Figure 3: Output from assertion failure ..............................................................................................8
Figure 4: Validation Method ................................................................................................................10
Figure 5: Structure of the project solution ..........................................................................................12
Figure 6: Method for automation of GUI tests .....................................................................................13
Figure 7: Dialog for creating a Site Constructor object ........................................................................15
Figure 8: Example of a Site hierarchy ................................................................................................16
Figure 9: A test method with a data source ..........................................................................................17
Figure 10: Setting a trigger column .....................................................................................................18
Figure 11: Creating a new list item .......................................................................................................18
Figure 12: Customized method ............................................................................................................19
Figure 13: Time web part .....................................................................................................................20
Figure 14: Comparison of implementation time ..................................................................................26
Figure 15: Comparison of runtime .....................................................................................................26
Figure 16: Cost comparison between manual and automated testing ...............................................27

Table 1: Initial time plan .......................................................................................................................11
Table 2: All selected test cases ............................................................................................................14
Table 3: Results for test cases from the Site constructor area .............................................................22
Table 4: ROI for test cases from the Site constructor area ....................................................................22
Table 5: Results for test cases from the military time code area .........................................................23
Table 6: ROI for test cases from the military time code area ...............................................................23
Table 7: Results for test cases from the time area ................................................................................23
Table 8: ROI for test cases from the time area .....................................................................................24
Table 9: Results for test cases from the copy list item area ..................................................................24
Table 10: ROI for test cases from the copy list item area ....................................................................25
1 Introduction

This report serves as an introduction to the thesis provided by the company Basalt AB. Explanations about common concepts, a problem definition and the goals of this thesis are described in section 1. Section 2 is focusing on related research to get a deeper understanding of the problem and to why this thesis is to be done. The method used for this thesis will be described in section 3. Section 4 will describe the implementation of automated testing. In section 5 the results will be presented and later on discussed in section 6. Conclusions will at last be drawn in section 7.

1.1 Concept explanation

Various concepts regarding this thesis is explained more in details in the following subcategories so that the problem definition will be easier to understand.

1.1.1 Software testing

Software testing is one of the essential processes in software development life cycles to find software bugs [1]. There are different methods available to do the software testing. One way is to create test cases to run simultaneously with the code that should be tested. Different test cases should together make sure that all possible paths in the code are tested [2].

1.1.2 Regression testing

Regression testing is done to check if changes to software programs have affected the existing features and functionality that has worked properly before [3]. This is usually done when a new version of a program is to be released, when components with new functionality have been added to the product or when other components that might affect the program have been altered [4].

1.1.3 GUI testing

The process of testing the functionality of a product’s graphical user interface (GUI) and making sure it works as expected is called GUI testing. This is done either manually or automatically with various test cases automated [5].

1.1.4 ROI

Return on investment (ROI), is a technique where calculations are used for deciding if the benefits is higher than the investments spent. [6]

1.1.5 Test Case

A test case is a documented step or sometimes a sequence of steps that describes an input, action or event to be tested. A test case also includes the expected response, to determine if the feature tested works correctly, due to behaviour and functionality. The documentation of a test case usually includes descriptions regarding prerequisites, input values, instructions on how to execute the test, expected results, name, ID, category, version, purpose, execution order, depth and author of the test case [7].
1.2 The company and SWECCIS
This thesis is made at a company called Basalt AB. What the company does is to deliver turnkey systems that are used for command and control within the Swedish armed forces. The systems are called SWECCIS. They are used both for national defence and exercises as well as in international missions where Sweden takes part. Some of the applications used in the SWECCIS system are developed by Basalt and some are standard components that are integrated. The applications developed are built using C#, .NET and Visual Studio 2013. A vast majority of these developed components are made for Microsoft SharePoint 2010. SWECCIS-systems are not like ordinary Windows systems, it is a much more complex system with advanced security features that is required to handle classified information in a military context. Systems are produced in a number of configurations to be able to handle different levels of classified information.

1.3 Tools
The tools used are described in the following subcategories.

1.3.1 Microsoft Office SharePoint Server (MOSS)
SharePoint is a platform that provides the possibility to build web-based applications and portals [8]. SharePoint portals can act as the host for many other web-based applications. Based on different forms of lists it works as a secure place to store and organize files. It also provides possibility to share information with other people. For example, widgets in form of shared workspaces can be added so that multiple people can work on a single document at the same time [9]. One of the main features of SWECCIS is a SharePoint portal and this is where the majority of testing will take part in the form of a simulation of a number of end users.

1.3.2 Coded UI
CodedUI is a tool introduced in Visual Studio 2010 Premium/Test Professional that focus on testing software systems through the graphical user interface (GUI). This framework can be used to test that the whole program is working correctly. CodedUI can be used to record actions manually made in the user interface and then the tool automatically generates code making it possible to automatically replay those actions and validate results. With this framework a lot of test scenarios can be prepared and then executed one after another [13].

1.3.2.1 Coded UI test builder
A recording is started through the Coded UI test builder as shown in Figure 1.

![Figure 1: Coded UI Test Builder](image)

The available actions that can be performed are to start/pause the recording, inspect all recorded actions and delete unwanted ones and then it’s possible to create a method of the recorded actions. When creating the method code is generated. You also get to name
that method and describe what it is about. The Coded UI Test Builder has the ability to create assertions as well. Assertions is described in section 1.3.2.3.

1.3.2.2 UI Maps
When a test case is created, code is generated presenting the actions made on the UI controls. This code is stored in a UI Map structured as shown in Figure 2. This map store all the objects of the Application under test (AUT) as a class file, which are then split into different files: UIMap.Designer.cs and UIMap.cs. All generated code will automatically be placed in the designer.cs file. This file include information about the components used during the recordings, it also includes code for performing the actions and if any data were entered it is stored as parameters in this class. This class cannot be manipulated because any changes made on the code will be lost if any new recordings are executed or changes to any existing recordings are done.

![UI Map structure](image)

1.3.2.3 Assertions
Assertions can be made to validate if the expected values or results actually are met. It can for example be to check if a control is visible or not, or if the output of something is correct. Assertions are added using the Coded UI Test Builder. This is done by first clicking the button that look like a circle shown in Figure 1 and then dragging the mouse pointer to the control that should contain the assertion. When that control is marked, the properties of the control will be visible. Assertions, like for example equal to or not equal to, can be added to the chosen property. The assertion will then compare the actual result with the
expected condition [17]. If the result of that is correct, no failure will occur. Otherwise an error message and a screenshot of the problem will appear as shown in Figure 3.

![Figure 3: Output from assertion failure](image)

### 1.4 Problem definition
Currently applications developed by Basalt are manually tested. The systems developed by the company need to be of great quality because if errors occur in the system and it ends badly it could affect the national security. Therefore testing is a crucial process. Because of the necessary level of security, some features in the system are restricted to a certain configuration of the system, so the functionality might differ significantly between them. Although many of the tests made are the same, they need to be done on all the different variations of the system. Huge amount of tests are therefore needed to establish the high level of quality the company aims for. Not only does the testing itself take time, but also the surrounding work such as planning, preparations of test environments, development and maintenance of test documentation are time consuming. Due to the dimensions of time and money that needs to be invested to carry out tests manually, the amounts of possible tests that can be carried out are limited. The following questions are answered in section 6 to solve the problems described.

1. To what extent can test automation with a tool like CodedUI, be used to replace testing of SharePoint functionality using manual labour?

2. Are automated testing with CodedUI the answer to improve the efficiency, regarding invested time, quality and cost during the testing process for the company’s current applications?

### 1.5 The goal
The goal for this thesis was to use the method presented in the methodology section below to investigate whether or not automated testing is worthwhile to implement at Basalt. To validate the investigation a proof of concept will be constructed that will show the automation of the selected test cases. The selected test cases will be some of the most important test cases required for a selected number of SharePoint components developed by the company. This proof of concept will make it possible for conclusions to be drawn
about whether it is possible or not to replace/reduce the manual labour of testing with automated testing. In addition to the conclusions, documentation of possible solutions should be made with recommendations and advice on how to best construct a suitable extendable framework that is easy to maintain. This should lead to the production of a well-documented report that explains the reasons for the given results and provides answers to the research questions provided in section 1.4. The report will also serve as a future guideline to use as a foundation if the automated testing done in this thesis would lead to further expanding.

2 Background

An overview of the background to this thesis and how the research front stands at the moment is given by a State of the art (SOTA) in section 2.1. The reason for this is to see the research area that this project revolves around from a wider perspective. The research areas included in this report are about software testing, automation of GUI testing. The following subcategories will explain some related research made in those fields.

2.1 State of the art

This state of the art will discuss the areas software testing and automation of GUI testing. A study made on automation of software testing explores factors affected by the use of automated testing. In this study automation of testing is described as “to include the development and execution of test scripts, verification of testing requirements, and the use of automated test tools”. Reasons for using automated testing is according to the study because manual testing takes time and automated testing can increase the efficiency. The benefits of automated testing due to this study includes “quality improvement through a better test coverage, and that more testing can be done in less time.” Disadvantages found were “high costs that include implementation, maintenance and training costs”. If maintenance of testing automation aren’t handled the cost for updating the entire thing can cost more than if the testing was done manually. So maintenance is of huge importance for automated testing to be worthwhile to use. Another important aspect for it to be worthwhile is the reusability of the testing automation system and test cases [10].

According to research found, automation of GUI tests is a difficult process to proceed with, and are often avoided. This is based on several reasons: “GUI testing requires some manual script writing”. The tool to be used in this thesis is called CodedUI, which is a tool that use the feature “capture replay” mentioned in the article, that means that the tool remember what the tester does to the GUI and then generates code from that. In the article that type of tool is mentioned as a great idea but that rarely works for automation of complete systems. The second reason is “the challenge of getting the tool to work with the product”. This is an issue that won’t be a problem in this thesis due to the fact that CodedUI already are proven to work with the company’s applications. But it might need some customizations to work with different features of the applications. A third reason mentioned in the report is “complication of keeping up with design changes made to a GUI”. Changes will often be made to the GUI to improve something, and to update the
automated test according to this can be very time consuming. The interface needs to be tested but according to recommendations found in the report, these tests should not include testing core functionality of the product. Those functions should be tested by other tests that don’t break if the GUI is redesigned [11]. Automated tests cannot replace manual tests completely [12].

This is just some research that gives good thoughts to start with before investigating further, especially regarding GUI testing. This thesis will if successful find solutions that makes GUI testing a bit easier than the SOTA declare.

3 Methodology
The method that has been used to reach the goals of this thesis is called the validation method. The focus of that method is on investigating the chance of finding the solution using CodedUI. Investigation has been carried out systematically including: experiments, simulations, mathematical analysis and proof of properties. Figure 4 show how this validation will be carried out in several steps.

1) The first step is to pick out test cases, approximately an amount around 10-15. After one round of this method, at the end of step 4 the same process will be made on another 10-15 test cases.
2) The second step will focus on learning everything about the test cases, what they are about, how they are carried out and what the expected outcome are.
3) The third step is where the test cases will be executed using CodedUI to see how the automated tests will occur. This is a step of work to see if it is even possible to accomplish automated testing. This will be made for all selected test cases. Problems that occur will be investigated so that solutions that work around them can be found. When no more problems exist a proof of concept will be created due to the solution provided.
4) The forth step is focused on providing answers and validate the research questions of this thesis. The chosen test cases has been executed manually before, so the results from that will be compared to the results of the same test cases after the automation. The comparison will be made on the factors: time and quality. These questions should therefore be answered.
   a. Is the time spent decreased?
   b. Is the functionality still the same?
5) This last step will be about documenting the results found in the previous step. The documentation will be in form of this final report.

![Figure 4: Validation Method](image-url)
The company already chose CodedUI based on their requirements and compatibility with their other applications. They wanted to find out if it was possible to use this tool in the solution of the thesis. That is why the method presented are most suitable in the given context. Due to the fact that the results of this method will show if the solution is valid or not. The validation method presented for this thesis is based on the fact that a tool already exist. It is predefined before the start of the project that CodedUI will be used. Therefor no energy will be spent on exploring the different tools available. Other methods without a pre-specified tool will have to go through a process like: investigating all suitable tools, compare them to each other and against the manual testing methods. When this is done, a well suited tool can be selected. So for these other methods the time consuming process of finding a tool needs to be done before the work towards automated testing can begin.

3.1 Initial time plan

The time period of the thesis work was roughly planned and is visible in the time plan shown in Table 1. The work of the project was divided in 5 steps as described in Figure 4 and the plan was to keep working on the report during the whole process.

<table>
<thead>
<tr>
<th>Week</th>
<th>Phase 1</th>
<th>Phase 2</th>
<th>Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Preparations</td>
<td>Writing the introduction and background</td>
<td>Updating the introduction and Background</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Updating the thesis report</td>
<td>Updating the thesis report</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Updating the thesis report</td>
<td>Updating the thesis report</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>Finalizing the thesis report</td>
<td>Finalizing the thesis report</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>Finalizing the thesis report</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>Submitting the Report</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Initial time plan

4 Implementation

Staff working at the Company have picked out the test cases selected for this industrial case study. This personnel often deals with the SharePoint portal of SWECCIS and associated test cases in general. The reason for this was to find test cases that if performed manually are very time consuming. That type of test cases often involves large entities of input values to enter or managing the site settings. That’s why the test cases shown in Table 2 would be very suitable for further investigation. To get a broad range of test cases, the chosen ones are of various kind and are performed in different ways from each other. To increase the perspective even further the test cases belongs to these four different areas of the SharePoint portal:

- **Site Constructor**: a tool that is used for creating site hierarchies.
- **Military Time Code**: a field type, used for showing date and time using a predefined format.
- **Time**: a web part, used to display any number of clocks with the same or different time zones. The clocks can be shown as digital or analogue.
- **Copy List Items**: a workflow that can be used to copy or move objects between lists.

The results of the selected test cases will be shown in separate tables in section 5, one for each of the areas described above.

Before starting to automate any tests a learning process was needed. Focus was on learning as much as possible about Coded UI and the selected test cases, but also to get familiar with the SharePoint portal where the testing would take place. To learn how to interact with and use Coded UI, watching tutorials and videos about the tool was helpful [14]. UI Maps was created, one for each type of the areas as shown in Figure 5.

![Figure 5: Structure of the project solution](image)

To learn about the test cases, the content of them had to be thoroughly read through. Next step was to locate where all the components and controls explained in the test case descriptions was situated in the SharePoint portal. This was done before manual testing could be executed. When things was unclear the supervisor at the company was helpful and explained more about the content. But to really understand the test cases and how they should be performed they were executed manually by hand. Then the results could be compared with the expected results, to see if the test would pass or not. This was a time consuming process, due to the amount of input values and things to compare. The time it took to execute the test cases manually was saved for later comparison with time spent on the automated tests. The next step regards the investigation of the possibility to use Coded UI to perform the testing. As shown in Table 2, 26 test cases was chosen for this
industrial case study and the method describing the investigation-process that was used on each and every one of them is shown in Figure 6. That process will be described more in detail in the sections below, where 5 test cases has been selected out of those 26 to exemplify the process towards automation. Things that have been helpful in some way and encountered during this process will be described in the sections “Lesson learned” for the test cases.
<table>
<thead>
<tr>
<th>Test area</th>
<th>Test case</th>
<th>Description of the functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site Constructor</td>
<td>TC200</td>
<td>Create a new Site Constructor object.</td>
</tr>
<tr>
<td></td>
<td>TC201</td>
<td>Create a new subnode.</td>
</tr>
<tr>
<td></td>
<td>TC202</td>
<td>Select another node.</td>
</tr>
<tr>
<td></td>
<td>TC203</td>
<td>Remove a node and all its subnodes.</td>
</tr>
<tr>
<td></td>
<td>TC204</td>
<td>Save a Site Constructor object.</td>
</tr>
<tr>
<td></td>
<td>TC205</td>
<td>Export a Site Constructor object as a XML file.</td>
</tr>
<tr>
<td></td>
<td>TC206</td>
<td>Import a Site Constructor object and upload the file content.</td>
</tr>
<tr>
<td></td>
<td>TC207</td>
<td>Create a site hierarchy using a Site Constructor object.</td>
</tr>
<tr>
<td>Military time code</td>
<td>Create Military Time Code Columns</td>
<td>Create new columns in a custom list according to the table shown in the test manual.</td>
</tr>
<tr>
<td></td>
<td>New list item</td>
<td>Create a new list item, and enter the information in the created columns as shown in the table in the test manual. Test that wrong time formats will generate error messages.</td>
</tr>
<tr>
<td></td>
<td>Correct the validation errors</td>
<td>Enter the information shown in the table in the test manual into the different time format fields. Test that no error messages occur when correct time formats are used.</td>
</tr>
<tr>
<td></td>
<td>Check Views</td>
<td>Check that all items in the custom list have the same values and formats in the viewform as in the editform.</td>
</tr>
<tr>
<td></td>
<td>Edit Time Zone</td>
<td>Change the time zone in Time Zone Settings to C for the columns: International and Swedish extended. The letter C represent a military time zone format.</td>
</tr>
<tr>
<td>Time</td>
<td>TC100</td>
<td>Remove all clocks from the web part and apply the change. The web part should still work but not show any clocks.</td>
</tr>
<tr>
<td></td>
<td>TC101</td>
<td>Change clock settings. The settings should be saved and the clock time zone, clock title and clock daylight saving settings should be changed.</td>
</tr>
<tr>
<td></td>
<td>TC103</td>
<td>Verify that both digital and analogue clocks work and are shown correctly. Change the Show analogue clocks setting for the web part.</td>
</tr>
<tr>
<td>Copy List Item</td>
<td>TC100</td>
<td>Change the workflow settings to use a field as a trigger column for copying the item to the destination list/library. Then create an item without marking that trigger. The item should not be copied. Check the status.</td>
</tr>
<tr>
<td></td>
<td>TC101</td>
<td>Change the workflow settings to use a field as a trigger column for copying the item to the destination list/library. Then create an item and marking the trigger. The item should be copied. Check the status.</td>
</tr>
<tr>
<td></td>
<td>TC102</td>
<td>Run the workflow twice with the same item. The item should not be overwritten at the destination list. Check the status.</td>
</tr>
<tr>
<td></td>
<td>TC103</td>
<td>Mark the Overwrite destination item option in workflow settings. Run the workflow twice with the same item (document). The document should be overwritten at the destination list. Check the status.</td>
</tr>
<tr>
<td></td>
<td>TC104</td>
<td>Enable Content approval on a list/library. Create a new item. Check that the item was not copied.</td>
</tr>
<tr>
<td></td>
<td>TC105</td>
<td>Enable Content approval on a list/library. Create a new item and approve it. Check that the item was copied.</td>
</tr>
<tr>
<td></td>
<td>TC106</td>
<td>Enable Require check out on a library. Create a new document. Check that the item was not copied.</td>
</tr>
<tr>
<td></td>
<td>TC107</td>
<td>Enable Require check out on a library. Create a new item and check in the document. Check that the item has been copied.</td>
</tr>
<tr>
<td></td>
<td>TC108</td>
<td>Mark the Move Item option in the workflow settings. Make sure that this workflow is used on the specific item created. The item is moved after being approved or checked in.</td>
</tr>
<tr>
<td></td>
<td>TC109</td>
<td>Verify that it is not possible to mark the Overwrite destination item option while the Move Item option is enabled.</td>
</tr>
</tbody>
</table>

Table 2: All selected test cases
4.1 Test case 1
This test case is called TC200 and is based on the creation of a new Site Constructor object, and therefore located in the Site constructor area. All the test cases are explained and the functionality of them is described in a test manual provided by the company. Clarifications on the expected results of a test are also provided in the manual, but that is all the information that is provided for the testing.

Figure 7: Dialog for creating a Site Constructor object

4.1.1 Automation process
For this test case the tester is supposed to click on a button called “New item”. The Coded UI Test Builder is used to record that click-action and a method will be generated. As described in the test manual the expected results is that a dialog-window should open up as shown in Figure 7. All the values presented in that dialog is expected to be the same as the default values described in the manual. To validate that, assertions is used. This is done by comparing the visible values with the values that are supposed to be there by default. The assertions will work as that comparer and check if those values are equal to each other, if not, an error message will appear. Assertion is also used for validating that the dialog is existing. If no assertion-failures is encountered during runtime, the test has passed.

4.1.2 Lessons learned
After the end of testing a clean-up should occur where things created during runtime of the tests are removed. If something was removed during testing, that should be added
again. Clean-up is important because errors can occur when running the tests repeatedly if items to be created already exists with the exact same information, it is not always possible to create multiple items of the same kind.

4.2 Test case 2
This test case is from the Site constructor area and is called TC203. For this test case a node in a site hierarchy is supposed to be removed. To be able to remove a node, it has to exist first. So to save time and effort it is therefore suitable for this test case to be executed after test case 1, where by default a node called “New operation” is created. The option to delete a node is not available at the first node in the hierarchy. To be able to perform this test case, at least one subnode have to be created, in this case called “New unit” as shown in Figure 8. The expected results for this test case are described like this in the manual:

“The node including all its subnodes are removed”.

![Figure 8: Example of a Site hierarchy](image)

To get full test coverage, the created subnode should have subnodes of its own. That is why the node called “New site” is also created. No matter what name the nodes possesses the output of this test case won’t change, so the default names of the nodes remain the same. Now a site hierarchy with a depth of three levels has been created. Even if not mentioned in the description, it is needed as a prerequisite to be able to meet the expected results completely after testing.

4.2.1 Automation process
At first the creation of the site hierarchy, described in the section above is recorded with the Coded UI test builder. Then deletion of the node “New unit” visible in Figure 8 is recorded. These two steps are generated as two separate methods to detach the prerequisites from the actual testing. To validate the tests, assertions are created on the removed node as well as on the associated subnodes. These assertions will check if the controls exists or not. If they are not, the assertions will pass without errors.

4.2.2 Lessons learned
This test case is relying on test case 1 so therefore the execution order is very important to avoid errors. A good thing can be to name the test methods in a numbered order so that they will be executed after each other. To be able to manage the order of test methods an ordered test should be created [16].

4.3 Test case 3
This test case is from the Military time code area and are called “New list item”. This test case is about creating a new list item with given values. When creating an item the tester gets to fill in information about the different available time formats: Swedish,
International, Swedish extended and Swedish no seconds. The values used are collected from a table available in the test case description in the test manual. Some of the values entered are expected to cause errors. So to pass the test, error messages must be visible and located where they are supposed to.

4.3.1 Automation process
At first a recording of all the values being entered in the fields of the different time formats was made. This process is saved and generated as a method. To validate the results assertions was added to check that the expected error messages was visible and situated at the correct location. When the method was generated, the values that was entered are hard coded into the method. This means a new recording would have to take place to use other values. Instead of that, the textboxes can be filled with data from a data file instead of the hard coded values. Figure 9 shows how to use a data connection for connecting to a .csv file that holds the values to enter. This is a great approach for testing with a large amount of input options or if many different combinations of input values are to be tested. New values to be tested can just be added to the .csv file or changes can be done to values already existing in the file.

![Figure 9: A test method with a data source](image)

4.3.2 Lessons learned
When using a data source in form of a .csv file in combination with Coded UI the following two settings has to be changed for it to work [15].

- In Visual studio change a property of the .csv file from Do not copy to Copy if newer.
- The .csv file has to be saved using the right encoding: Unicode (UTF-8 without signature) – Codepage 65001

To customize a generated method, the method has to be moved into the .cs file located in the UI Map. The option to move the code are built in to Coded UI and is easy to manage. If not moved, any changes to the code will be lost each time a new recording is made. So move first and then customize the method as much needed. After making changes, it is important to remember to check if the method still works as expected. When the method has been moved, built in functions like removing steps, split methods and renaming is lost
and cannot be used on that method any longer. So Coded UI does not need to be used by someone with developer knowledge until customizations should be made.

4.4 Test case 4
This test case is called TC100 and are from the Copy list item area. The test case is divided in two parts. As Figure 10 show, the first part is about changing the workflow settings, so that a trigger column is used for deciding if the item should be copied or not.

![Figure 10: Setting a trigger column](image1)

As seen in Figure 11 the second part is about creating an item without marking the checkbox “Copy” represented as the trigger. The item should therefore not be copied. The item is created in a list situated on the third level in the site hierarchy. If the item were to be copied, it would be copied to the first level in the hierarchy. To pass this test the item should have the status: “Copy condition not met” and the item should not be copied and therefore not exist in the list on the first level.

4.4.1 Automation process
At first the whole test case was recorded and generated into a single method. Later on the actions was re-recorded and divided into two parts to make it easier to see if the test was working correctly. When doing this a few things was discovered. The status was not always visible, sometimes it was calculated when the page was refreshed, so the status could be: “Starting” or “In progress”. To meet all of these conditions each scenario was recorded separately and generated into methods so that information about the controls
could be received. As the test cases will run sequentially an issue was encountered because of the different conditions. If the wrong method was running and the condition was different from the actions recorded and generated for the method, the test would fail. By moving the methods created for the conditions to the .cs-file the code could be customized and merged into just one method as shown in Figure 12. Conditions can be used inside a method if customized but that is not something that is supported to do automatically using the Coded UI Test Builder. After trying the conditions an assertion will be used to validate if the expected result is met. Afterwards the item created is deleted to enable the possibility to run the test again without encountering any errors based on a previously made test.

4.5 Test case 5

This test case is from the Time area and is called TC101. The aim for this test case is to change clock settings for the web part. The test manual does not exactly describe what kind of settings to change, but the expected results are described as the following:

“All changes have been saved (if the web part settings is reopened). The clock time zone, the clock title and the clock daylight saving has changed.”

To be able to reach the expected results the above description is important and will act as further directions on which settings to change, which now will be: the time zone, title and daylight savings for the clock.
4.5.1 Automation process
If no clocks are visible at the site, some will be created before the start of the test as in Figure 13. The next step is to create assertions to save the settings before changing them. Then, after recording the actions, assertions will again be created and compared against the previous results. If the results are different from each other, the changes has been made and the test has passed.

![Figure 13: Time web part](image)

4.5.2 Lessons learned
When working with assertions there are some techniques that can be useful. The obvious way to use them is to for example to add an assertion when the correct answer is visible or a control is existing on the screen. The assertion will then fail if the same correct state is not reached during the runtime of the test. Another way to use assertions is to add assertions to values or controls that will be used later, in that case the information is saved and generated as code. That code can be customized and used with other created assertions later on. The values can also be changed in the code and not just by using the Coded UI Test Builder.

5 Results
The measured times for the selected GUI test cases are presented in the tables in the following sections. There is one table for each affected test area of SharePoint. The times are measured for the implementation as well for runtime of the tests. The manual implementation time estimates how much time that was spent when creating the test cases, descriptions and expected results for the test manual. The implementation time for the automation measure how long it took to use the information in the test manual and then automate those GUI tests until they were running smoothly. If just the time it takes to click and enter values are measured as the runtime for the test cases, those tests would not be executed as thoroughly as the automated tests is. To be able to compare the values properly, they should have the same standard. Doing the manual testing thoroughly will take longer and that is what the values shown in the tables stands for. The times specified for the manual tests were estimated by the test manager together with testers at the company and are not based on own measurements. The automated runtimes are calculated automatically by Coded UI when running the created test-methods.
5.1 ROI

For a complete calculation of the Return on Investment (ROI) these factors should be included: the cost of the tool to be used for the solution, the cost for automating the tests in form of time and the time it costs for maintaining the automated tests [6]. Some of these things are not relevant for this industrial case study and will not be part of the calculation of ROI. That has to do with the fact that a tool was already chosen and in use before the starting point of the thesis work. So costs regarding the test tool (such as evaluation, licenses and the procurement of Visual Studio and Coded UI) will be dismissed. There were no possibility to calculate costs for maintenance of the test cases during the period of this thesis so that factor will have to be dismissed as well.

How often the automated tests will be executed should be considered when calculating the benefits, which are often measured by the things we save by using automation. That can take the form of savings in costs and time, increased efficiency, increased quality of the product and reduced risks. The three following formulas will be used for the calculation of the time-savings of ROI and calculate the brake-even point $E$ [6].

\[
\begin{align*}
(1) \quad ROI_{\text{automated}} &= V_a + n \times D_a \\
(2) \quad ROI_{\text{manual}} &= V_m + n \times D_m \\
(3) \quad E(n) &= \frac{ROI_{\text{automated}}}{ROI_{\text{manual}}} = \frac{V_a + n \times D_a}{V_m + n \times D_m}
\end{align*}
\]

As seen in formula (1) $V_a$ stands for the implementation time of the automated test case, $D_a$ stands for the automated runtime and $n$ stands for the number of executed test cases. The same regards formula (2) but the times are calculated for executing the test cases manually. To be able to see when automated testing will have been repaid formula (3) is used to compare automated testing to manual testing and find the break-even point between the two approaches. The Return on Investment of time saved has been calculated for each test case and the results are presented in the sections described by area beneath. Quality, efficiency and reduced risks will be estimated according to the results from the calculations and will be discussed in section 6, the importance of the results from the calculations will also be discussed in that section. Formula (4), (5), (6) describes how the ROI was calculated for each test case [18].

\[
\begin{align*}
(4) \quad ROI_{\text{Testcase}} &= \frac{Benefits}{Costs} \times 100 \\
(5) \quad Benefits &= Runtime_{\text{manual}} - Runtime_{\text{automated}} \\
(6) \quad Costs &= Runtime_{\text{manual}}
\end{align*}
\]
5.2 Site constructor
Table 3 describes the resulting time periods spent on implementing the test cases and executing them. The table describes the differences between doing the testing manually compared with doing that automated. The implementation time differs between the two approaches by 1-2 hours. This means that the automation approach takes a bit longer to implement but each round of test runtime is much faster. Test rounds using automation take less than 3 minutes compared to the manual runtime that take over 2 hours to complete. Table 4 shows the calculations of ROI for test cases of the Site constructor area.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Manual Implementation-time</th>
<th>Automation Implementation-time</th>
<th>Manual runtime</th>
<th>Automation runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>1h</td>
<td>5 min</td>
<td>1 min</td>
<td>10 sec</td>
</tr>
<tr>
<td>TC200 - new SC-object</td>
<td>10 -15 min per test case</td>
<td>20 min per test case</td>
<td>10 min</td>
<td>21 sec</td>
</tr>
<tr>
<td>TC201 - new subnode</td>
<td>30 min</td>
<td>15 min</td>
<td>9 sec</td>
<td></td>
</tr>
<tr>
<td>TC202 - select node</td>
<td>30 min</td>
<td>15 min</td>
<td>5 sec</td>
<td></td>
</tr>
<tr>
<td>TC203 - remove node</td>
<td>10 min</td>
<td>10 min</td>
<td>20 min</td>
<td>6 sec</td>
</tr>
<tr>
<td>TC204 - save a SC-object</td>
<td>10 min</td>
<td>10 min</td>
<td>8 sec</td>
<td></td>
</tr>
<tr>
<td>TC205 - export a SC-object</td>
<td>10 min</td>
<td>10 min</td>
<td>20 sec</td>
<td></td>
</tr>
<tr>
<td>TC206 - import SC-object</td>
<td>15 min</td>
<td>15 min</td>
<td>25 sec</td>
<td></td>
</tr>
<tr>
<td>TC207 - create site hierarchy</td>
<td>30 min</td>
<td>25 min</td>
<td>11 sec</td>
<td></td>
</tr>
<tr>
<td>Clean up</td>
<td>-</td>
<td>1h</td>
<td>20 min</td>
<td>27 sec</td>
</tr>
<tr>
<td><strong>Total time:</strong></td>
<td><strong>140-180 min</strong></td>
<td><strong>255 min</strong></td>
<td><strong>141 min</strong></td>
<td><strong>2,37 min</strong></td>
</tr>
</tbody>
</table>

Table 3: Results for test cases from the Site constructor area

<table>
<thead>
<tr>
<th>Test case</th>
<th>Investment (minutes)</th>
<th>Benefits (minutes)</th>
<th>ROI (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC200 - new SC-object</td>
<td>15</td>
<td>15 - 0.15 = 14.85</td>
<td>(14.85 /15) * 100 = 99</td>
</tr>
<tr>
<td>TC201 - new subnode</td>
<td>15</td>
<td>15 - 0.08 = 14.92</td>
<td>(14.92 /15) * 100 = 99,5</td>
</tr>
<tr>
<td>TC202 - select node</td>
<td>20</td>
<td>20 - 0.1 = 19.9</td>
<td>(19.9 /20) * 100 = 99,5</td>
</tr>
<tr>
<td>TC203 - remove node</td>
<td>10</td>
<td>10 - 0.35 = 9.65</td>
<td>(9.65 /10) * 100 = 96,5</td>
</tr>
<tr>
<td>TC204 - save a SC-object</td>
<td>10</td>
<td>10 - 0.13 = 9.87</td>
<td>(9.87 /10) * 100 = 98,7</td>
</tr>
<tr>
<td>TC205 - export a SC-object</td>
<td>10</td>
<td>10 - 0.33 = 9.67</td>
<td>(9.67 /10) * 100 = 96,7</td>
</tr>
<tr>
<td>TC206 - import SC-object</td>
<td>15</td>
<td>15 - 0.42 = 14.58</td>
<td>(14.58 /15) * 100 = 97,2</td>
</tr>
<tr>
<td>TC207 - create site hierarchy</td>
<td>25</td>
<td>25 - 0.18 = 24.82</td>
<td>(24.82 /25) * 100 = 99,28</td>
</tr>
</tbody>
</table>

Table 4: ROI for test cases from the Site constructor area
5.3 Military time code

Test cases that belong to the military time code field type often include a large amount of data do enter, that is noticeable at the implementation for the automated testing on the three first test cases shown in Table 5. Table 4 shows the resulting time spent for both implementation and runtime.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Manual Implementation time</th>
<th>Automation Implementation time</th>
<th>Manual runtime</th>
<th>Automation runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>30 min</td>
<td>1h</td>
<td>10 min</td>
<td>25 sec</td>
</tr>
<tr>
<td>Create Columns</td>
<td></td>
<td>2h per test case</td>
<td>10 min</td>
<td>37 sec</td>
</tr>
<tr>
<td>New list item</td>
<td>15 – 20 min per test case</td>
<td></td>
<td>20 min</td>
<td>29 sec</td>
</tr>
<tr>
<td>Correct the validation errors</td>
<td></td>
<td></td>
<td>15 min</td>
<td>29 sec</td>
</tr>
<tr>
<td>Check views</td>
<td>5 min</td>
<td>20 min per test case</td>
<td>10 min</td>
<td>11 sec</td>
</tr>
<tr>
<td>Edit time Zone</td>
<td>10 min</td>
<td></td>
<td>5 min</td>
<td>14 sec</td>
</tr>
<tr>
<td>Clean up</td>
<td>-</td>
<td>10 min</td>
<td>5 min</td>
<td>5 sec</td>
</tr>
<tr>
<td><strong>Total time:</strong></td>
<td><strong>90-105 min</strong></td>
<td><strong>470 min</strong></td>
<td><strong>75 min</strong></td>
<td><strong>2,5 min</strong></td>
</tr>
</tbody>
</table>

Table 5: Results for test cases from the military time code area

The implementation process of the automated tests took much longer than the manual one, in fact the difference is over 6 hours. The manual runtime is slower than the automated runtime, where the manual approach takes 75 minutes to complete and the automated only 2.5 minutes. Table 6 show calculations of ROI made on the test cases from the Military time code area.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Investment (minutes)</th>
<th>Benefits (minutes)</th>
<th>ROI (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC200 - new SC-object</td>
<td>10</td>
<td>10 - 0.62 = 9.38</td>
<td>(9.38/10) * 100 = 93.8</td>
</tr>
<tr>
<td>TC201 - new subnode</td>
<td>20</td>
<td>20 - 0.48 = 19.52</td>
<td>(19.52/20) * 100 = 97.6</td>
</tr>
<tr>
<td>TC202 - select node</td>
<td>15</td>
<td>15 - 0.48 = 14.52</td>
<td>(14.52/15) * 100 = 96.8</td>
</tr>
<tr>
<td>TC203 - remove node</td>
<td>10</td>
<td>10 - 0.18 = 9.82</td>
<td>(9.82/10) * 100 = 98.2</td>
</tr>
<tr>
<td>TC204 - save a SC-object</td>
<td>5</td>
<td>5 - 0.23 = 9.87</td>
<td>(9.87/5) * 100 = 197.4</td>
</tr>
</tbody>
</table>

Table 6: ROI for test cases from the military time code area

5.4 Time

Even though the test cases from the area time are few, the results are still quite clear. As seen in Table 7 the implementation of the automated tests takes more than an hour longer to create. The manual runtime results take almost an hour longer to complete than the automated one. The clean-up is almost the same for both the approaches. The automated runtime will be almost 50 minutes faster, but the implementation time is slower.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Manual Implementation time</th>
<th>Automation Implementation time</th>
<th>Manual runtime</th>
<th>Automation runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>5 min</td>
<td>10 min</td>
<td>10 min</td>
<td>25 sec</td>
</tr>
<tr>
<td>TC100 - remove all clocks</td>
<td>10 min per test case</td>
<td>15 min</td>
<td>10 min</td>
<td>17 sec</td>
</tr>
<tr>
<td>TC101 - change clock settings</td>
<td>10 min per test case</td>
<td>1h</td>
<td>20 min</td>
<td>11 sec</td>
</tr>
<tr>
<td>TC103 - verify that clocks works</td>
<td></td>
<td>15 min</td>
<td>10 min</td>
<td>28 sec</td>
</tr>
<tr>
<td>Clean up</td>
<td>-</td>
<td>10 min</td>
<td>1 min</td>
<td>30 sec</td>
</tr>
<tr>
<td><strong>Total time:</strong></td>
<td><strong>35 min</strong></td>
<td><strong>110 min</strong></td>
<td><strong>51 min</strong></td>
<td><strong>1,85 min</strong></td>
</tr>
</tbody>
</table>

Table 7: Results for test cases from the time area
Table 8 shows the calculations of ROI on individual test cases.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Investment (minutes)</th>
<th>Benefits (minutes)</th>
<th>ROI (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC200 - new SC-object</td>
<td>10 min</td>
<td>10 - 0,28 = 9,72</td>
<td>(9,72/10) * 100 = 97,2</td>
</tr>
<tr>
<td>TC201 - new subnode</td>
<td>20 min</td>
<td>20 - 0,18 = 19,82</td>
<td>(19,82/20) * 100 = 99,1</td>
</tr>
<tr>
<td>TC202 - select node</td>
<td>10 min</td>
<td>10 - 0,47 = 9,53</td>
<td>(9,53/10) * 100 = 95</td>
</tr>
</tbody>
</table>

Table 8: ROI for test cases from the time area

5.5 Copy list item

The test cases marked with (*) in Table 9 has automation runtimes that should be divided by two. That is because these test cases are run twice, one for lists and one for document libraries. The values presented are from the two test types, but the functionality is the same and therefore the time is calculated as one average value instead of two separate.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Manual Implementation-time</th>
<th>Automation Implementation-time</th>
<th>Manual runtime</th>
<th>Automation runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>1h</td>
<td>3h</td>
<td>30 min</td>
<td>3 min</td>
</tr>
<tr>
<td>TC100 - no trigger, check status</td>
<td>2h per test case</td>
<td>15 min</td>
<td>80 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC101 - trigger, check status</td>
<td>2h per test case</td>
<td>20 min</td>
<td>62 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC102 - run workflow twice</td>
<td>1h per test case</td>
<td>15 min</td>
<td>72 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC103 - overwrite destination item</td>
<td>10 – 15 min per test case</td>
<td>20 min</td>
<td>35 sec</td>
<td></td>
</tr>
<tr>
<td>TC104 - enable content approval</td>
<td>1h per test case</td>
<td>15 min</td>
<td>64 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC105 - enable content approval, and approve</td>
<td>2h per test case</td>
<td>20 min</td>
<td>84 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC106 - enable require check out, create document</td>
<td>30 min per test case</td>
<td>10 min</td>
<td>28 sec</td>
<td></td>
</tr>
<tr>
<td>TC107 - enable require check out, create document, check</td>
<td>30 min per test case</td>
<td>15 min</td>
<td>33 sec</td>
<td></td>
</tr>
<tr>
<td>TC108 - move item option in workflow</td>
<td>45 min per test case</td>
<td>20 min</td>
<td>80 sec (*)</td>
<td></td>
</tr>
<tr>
<td>TC109 - verify unable to mark other options</td>
<td>2h per test case</td>
<td>15 min</td>
<td>24 sec (*)</td>
<td></td>
</tr>
<tr>
<td>Clean up</td>
<td>-</td>
<td>2h</td>
<td>15 min</td>
<td>42 sec</td>
</tr>
<tr>
<td>Total time:</td>
<td>160-210 min</td>
<td>990 min</td>
<td>210 min</td>
<td>13 min</td>
</tr>
</tbody>
</table>

Table 9: Results for test cases from the copy list item area
5.6 Limitations

Due to the limited time of the thesis all test cases currently done manually on the SharePoint portal could not be investigated and automated. That is why only 26 test cases was picked out to be part of this research. Due to the high security, some confidential material can occur. If some of that information is found to have a significant importance for the thesis, a discussion with the company will be held to work around that problem. Some issues that occurred during this project can limit the work and create errors, for example a bad internet connection that cause loading of elements to fail. If this happen during runtime of a test, the long waiting time can force the program to believe that the controls it is waiting for is non-existing. Another problem caused by long waiting times is that if the Coded UI Test Builder is working slowly, then it sometimes misses actions made on the screen. If the person recording the test gets frustrated and for example clicks more than once on a button the test builder records both of those actions. To work around that problem, the tester should check the recorded steps before generating the code. If any actions are missing, they can be added and no problems will occur later on. If a specific step requires a long time for the system to respond a delay between the actions can be manually added to the code. This forces the test method to wait instead of failing due to the long waiting time. Such a wait is coded via the statement below. The number “1000” stands for how long the method should wait. This number can be changed to whatever is needed.

```
Playback.PlaybackSettings.DelayBetweenActions = 1000;
```

6 Discussion

The expected result of this thesis work is a proof of concept with arguments concerning whether or not the Return on Investment (ROI) is large enough for it to be worthwhile for
the company to continue on with the work. In addition the following research questions stated for this thesis work will be answered in this section:

(1) To what extent can test automation with a tool like CodedUI be used to replace testing of SharePoint functionality using manual labour?

(2) Are automated testing with CodedUI the answer to improve the efficiency, regarding invested time, quality and cost during the testing process for the company’s current applications?

The investment is for sure higher for the automated GUI testing with an implementation time of around 30 hours, compared to 7-9 hours that are spent doing the manual implementation. But to compare them like that can be misleading. The runtime is much more interesting to compare. To carry out the manual tests equally careful as the computer executes the automated, additional time is needed to check that all parts executes as expected. Automated testing will do exactly the same thing every time the test is run. If the test methods have been developed in a way that make the tests run smoothly and still check everything the quality of the tests will be higher than if doing the tests manually. Because even if a tester manually tries to run the tests in the same way each time, human mistakes can happen. What if the initialization and prerequisites aren’t made exactly the same way each time? Changes like that can cause a lot of problems. In some ways it might be good, because other problems can be encountered and fixed. It is however more likely that this kind of issues disturbs the execution of the test, rather than that the found problems actually is errors in the system.

Due to the amount of time it takes to execute the test cases, they will not be tested very often. What if some errors don’t occur until after being performed multiple times? That is hard to detect manually. If this instead would be tested automatically, the test cases could be executed over and over again, until it breaks or are stopped. That way unknown errors can be found and corrected in time, before the system reaches the user.

![Figure 14: Comparison of runtime](image1)

![Figure 14: Comparison of implementation time](image2)
Manually these 26 tests are estimated to take about 8 hours to do thoroughly. In that time preparations before the start of the testing is included but no clean-up afterwards. The automation of these test cases has narrowed the runtime down to about 33 minutes. That is a massive change, and even though the implementation take over 30 hours. As an example Figure 15 show the difference in executing the tests five times. The runtime for performing the tests manually one time is longer than performing the automated tests five times. Five rounds of testing will take 40 hours to do manually, to do it automated the implementation time of 30 hours and the runtime of 33 minutes x 5 will equal a total time of 32.75 hours. That show that the time spent on the implementation of the automated testing will be repaid before the tests have been executed 5 times. When the tests have been automated the tests can be executed over and over again with much less effort. If the company continues to do the testing manually, they will have to spend the same amount of time every time and it will never decrease. When the automated tests are created and the methods are done they can be used by anyone, no developer skills are necessary.

![Cost comparison](image)

Figure 16 shows how the cost outcome over time for the test automation is estimated to turn out compared to a manual approach. The cost of the investment is often high and time consuming at first, often greater than the labour of manual testing. But when the implementation is done, the cost will drop drastically and with time be very low. In comparison with manual testing when the same effort is spent every time, the cost will linearly get higher with time. For this industrial case study, the first time of executing the tests will take 30.55h for the automated testing and 15h for the manual testing. In these measurement the implementation time and the runtime is included. The second time the
tests are run the implementation time won’t be necessary for either of the approaches because it has already been developed and can be used repeatedly. So for that testing and all of the following test rounds after that will only be measuring the runtime of the tests. Figure 16 shows the total time spent as the cost for the approach. Values for the maintenance of testing could not be measured and are therefore not part of this calculation. After running the tests 20 times, 167 hours would have been spent executing them manually, and only 41 hours for automated testing. So after 20 test rounds 126 hours of manual testing would be saved if automated testing were used instead. The following calculations show that the break-even point, i.e. where the automated testing will be repaid for these 26 test cases, will be reached between test round three and four. This can be seen graphically in Figure 16, where the lines cross each other.

\[
E(n) = \frac{V_a + n \cdot D_a}{V_m + n \cdot D_m} = \frac{30 + n \cdot 0,55}{7 + n \cdot 8}
\]

\[
E(1) = \frac{30 + 1 \cdot 0,55}{7 + 1 \cdot 8} = \frac{30,55}{15} = 2,04
\]

\[
E(2) = \frac{30 + 2 \cdot 0,55}{7 + 2 \cdot 8} = \frac{31,1}{23} = 1,35
\]

\[
E(3) = \frac{30 + 3 \cdot 0,55}{7 + 3 \cdot 8} = \frac{31,65}{31} = 1,02
\]

\[
E(4) = \frac{30 + 4 \cdot 0,55}{7 + 4 \cdot 8} = \frac{32,2}{39} = 0,83
\]

Coded UI have great potential for replacing manual testing of SharePoint for Basalt AB. The tool is already used within the company. The licenses that are needed already exist, so an investment on that point will not be required. For the level of knowledge needed, it depends on the type of test case to automate. If the test case is built to just click buttons and validate outputs, the Coded UI Test Builder will help a lot, all that is needed is a short introduction on how to use the tool. When test cases include huge amount of data to enter, data driven testing with Coded UI would be very suitable. In that case data-files could be filled in with all the information that is to be entered, then one test method can be created that enters values according to the data-file. The test would be looping using one row of the file at the time, from the first row to the last. So each row would represent one item of information to verify. If conditions are needed for the test case, a developer or someone with some coding knowledge will have to customize the created methods according to the needs. That is not something that Coded UI supports to do automatically.

The test cases used in this industrial case study is only a small part of the test cases presented in the test manual provided by the company. Even if only 4 chapters out of 15 have been automated, the runtime has proven to become very fast and the cost of automation will be repaid fast. So there should be no doubt about whether or not it would be worth it to continue on with the work of automating tests.
The ROI calculations done on each test case show that all of them would have a benefit of more than 90%, in terms of time from automated GUI testing compared to the manual GUI testing.

### 6.1 Maintenance and guideline

No developing knowledge is necessary for using Coded UI, because of the simple record and playback functionality that will generate all code for the actions made. If any customization of that code are needed some developing knowledge would be good. To easily keep track of all the tests, use separate UI Maps for different parts of the application to be tested. That will also minimize the risk that changes to the recorded tests will affect other parts of the application. It will be easier to find and make changes to specific parts if the generated methods only consist of a few steps. It is better to use a lot of methods with few steps than few methods with a lot of steps included. Let's say one button in the application has been removed and another one has to be used instead. If an existing method only includes the action of clicking the button, it is easy to just record the same action with the Coded UI Test Builder again with the new button and then generate a method with the same name as the previous one. Then that method will be replaced and no other changes has to be made to the test suite. To avoid that this is happening when the GUI is redesigned, it is good if the developers just move the existing components instead of deleting them and creating new ones. If they are reused they keep the same id (which is used to connect and find the controls) and there is no need for changes to occur in the test methods. When errors are encountered Coded UI is built to save that information for each test case. A folder will include the logged information combined with screenshots if that is needed. For example if some dialog is on front of a control to be used, i.e. blocking it, a screenshot will show that dialog. This is a great way to learn about the problems, and correct them.

It would be very good if each test case had its own test method, because then the measured time would show how long just that one test took. Also, if something does fail, the entire test method will break. If it only contains one test case, it is easy to know which one that needs to be fixed. And the other test cases can continue on with the testing. A great way to save time and effort is to reuse methods. The actions that are performed often should be generated as methods with a few steps in it that only regards those actions. It can be just general actions as opening menus or go into settings. This will save a lot of time when implementing. More advices on things to think about can be found in the subsections in section 4.

### 7. Conclusions

Conclusions that can be drawn from this thesis work is that Coded UI most likely would be suitable to use for the test automation process at Basalt AB. Coded UI was very easy to use and if the advices given in this report are followed maintenance of the test suite would not be that time consuming. The calculations on return on investment was very positive and show that all of them would have a benefit of more than 90%, in terms of time saved from using automated GUI testing compared to the manual GUI testing. It would only take
3-4 times before the investment of automation would be repaid. So the fact that the time savings are so high and that the investment will be repaid so fast will solve many of the company’s testing problems, as being able to run tests more frequently with a high level of quality. The tests will be run in the same way each time so that will give higher quality to regression testing. So to round this up, in this situation automated testing definitely is worth the investment.

7.1 Future work
What if the tests can be run in parallel? Can input be entered in parallel instead of sequentially as it works today? These are questions that could be investigated in the future to see if it is possible to speed up the automated GUI testing even further and possibly use the test cases recorded for performance testing as well, simulating many users working in parallel. Researching a possibility to create some kind of script so that Visual Studio don’t have to be used when testing, Further calculations on ROI can be made in the future that includes measurements regarding other factors, such as the time it take to maintain the testing suite.
References


