AUTOMATING REGRESSION TESTING IN A DOMAIN-SPECIFIC SAAS SETTING

Oskar Renefalk
ork19001@student.mdh.se

Examiner: Wasif Afzal
Mälardalen University, Västerås, Sweden

Supervisor: Eduard Paul Enoiu
Mälardalen University, Västerås, Sweden

29/06/2023
Definitions

The following abbreviations are used in the thesis.

- SUT - Software under test
- SaaS - Software-as-a-service
Abstract

This thesis aims to provide insights into how implementing an automated solution for regression testing in a domain-specific SaaS environment can address common regression testing challenges such as high cost or poor test result visualization. Additionally, it presents a practical demonstration of how to approach these challenges and the potential development of the solution. While previous studies on test automation often recommend adopting a third-party automation tool to tackle certain regression testing challenges, this thesis takes a different approach. We explore the feasibility of developing a test automation framework specifically tailored for a SaaS company operating in Sweden’s insurance and pension industry. This industry is renowned for its complex infrastructure, and the company requires efficient testing to support frequent releases and accommodate its diverse range of customers.

To identify domain-specific issues and answer the feasibility of automating regression testing in a domain-specific setting, this thesis begins with a literature review and interviews to identify generic and company-specific issues related to regression testing and test automation. After identifying these issues, the thesis proceeds through the development process of a test framework, test execution, and framework analysis. It also explores the growing interest in test automation and examines the ethical considerations associated with its implementation.

Finally, the thesis provides a test automation framework development process starting with identifying the specific issues related to developing an automated solution for regression testing for a SaaS system, continuing with its development, and confirming the feasibility of creating such.
# Contents

1. Introduction ................................................. 1

2. Background ................................................ 2
   2.1 Essential Background and Terminology .................. 2
       2.1.1 Software as a Service (SaaS) .................. 2
       2.1.2 Regression Testing ............................ 2
       2.1.3 Test automation ............................... 2
       2.1.4 Domain Specific SaaS ......................... 2
   2.2 Regression Testing in Domain-Specific SaaS ............ 3
   2.3 Exploring the Benefits and Limitations of Test Automation .. 3

3. Problem Formulation ....................................... 4

4. Research Methodology ...................................... 5
   4.1 Data Collection ..................................... 7
       4.1.1 Literature Review ............................ 7
       4.1.2 Interviews ................................... 7
   4.2 Interview Design ................................... 7
   4.3 Data Analysis ....................................... 8

5. Interview & Literature Review Results ...................... 9
   5.1 Interview Results .................................. 9
   5.2 Literature Review Results .......................... 10
   5.3 Recognized Themes .................................. 11

6. Development of the Test Automation Framework ............ 13
   6.1 System Architecture .................................. 13
   6.2 SUT Limitations and Recommendations .................. 14
   6.3 Test Case Model ..................................... 15
   6.4 Test Framework Design ................................ 15
       6.4.1 Input Handling ............................... 16
       6.4.2 Result File Creation ......................... 16
       6.4.3 Result Validation ............................ 17
   6.5 Test Execution ..................................... 18
       6.5.1 Postman ....................................... 19
       6.5.2 Test Framework - Order of events ............ 19

7. Discussion ................................................. 22
   7.1 Test Framework Development Discussion ................ 22
   7.2 RQ1 Revisited - Identified Challenges ................. 22
   7.3 RQ2 Revisited - Feasibility Of Automating Regression Testing For SaaS Domain-Specific Systems ...... 23
   7.4 RQ3 Revisited - Development Of a Test Automation Solution for Regression Testing ............ 23
   7.5 Validity Threats ..................................... 23

8. Ethical and Societal Effects of Test Automation ........... 24
   8.1 Privacy and Confidential Data ........................ 24
   8.2 Automation Bias and Accountability ................... 24
   8.3 Social and Human Impact ............................. 24

9. Conclusions ................................................ 26
   9.1 Future Work ......................................... 26

References .................................................. 28
# List of Figures

1. Original Technology Transfer Model ........................................... 5
2. The modified technology transfer that aligns with the specific need of this thesis 6
3. Test Automation Architecture Overview ...................................... 14
4. Data Exchange Overview .......................................................... 20
5. JSON file containing all reported differences from a specific test case .......... 21
List of Tables

1. Steps in the problem-solving process ........................................ 6
2. Interview design ................................................................. 8
3. Interviewees’ experience ......................................................... 9
4. Success factors and challenges in test automation ......................... 10
5. Summary of the findings from previous studies ............................. 12
6. Main themes of the interview ................................................... 13
1. Introduction

Regression testing is one of the most crucial components of the software testing cycle, yet it is also the most time-consuming. Regression testing aims to verify that the software under test works the same way after code changes and to ensure that the code changes did not introduce unexpected bugs. Often, regression testing is carried out by re-running the entire test suite to provide complete test coverage. However, since the test suite is usually quite large, executing all the tests is a highly time-consuming process [1].

Today’s constant chase for cost-effectiveness in software testing has increased the interest in decreasing the time consumption of regression testing. One way of doing so is to automate the execution of the test suite. This is regularly done by purchasing a solution for automation from a third-party vendor. While this might appear as a straightforward solution, the complexity of doing so increases substantially when the SUT derives from a particular domain with unique peculiarities [2].

In this thesis, we identify challenges to regression testing in a domain-specific SaaS setting and propose a test framework that operates within a specific SaaS domain, specifically the Swedish insurance and pension industry. The thesis is performed in a company that encounters significant challenges in its endeavor to automate regression testing. Despite the availability of third-party vendors offering automated testing solutions, the idiosyncrasies within the company’s domain complicate the implementation of such a solution. This predicament is not exclusive to the company and has been reported in multiple regression testing studies, highlighting this thesis’s significance.

This thesis is organized into three main research questions. Firstly, it investigates the challenges faced by the company as it strives to implement an automated regression testing solution. Secondly, it explores the feasibility of automating the regression testing process for domain-specific SaaS systems. Finally, it examines how a test automation solution can be developed to tackle the regression testing challenges specific to domain-specific SaaS systems. By addressing these research questions, this thesis aims to provide insights and recommendations for effectively implementing regression testing in the context of specialized SaaS domains.

Using a modified version of Gorschek’s information transfer model [3], this research addresses the questions through five key steps: problem identification, problem formulation, solution development, solution review, and process analysis. By identifying domain-specific challenges, formulating straightforward research questions, developing tailored solutions, evaluating their effectiveness, and analyzing the overall process, this study aims to provide valuable insights and practical guidance for addressing the specific domain’s regression testing challenges. Using this method, we aim to thoroughly research and identify the complexities of automating regression testing in a domain-specific SaaS setting.

The thesis identifies several challenges faced in regression testing in a domain-specific SaaS setting. It contributes insights into the feasibility of developing an automated solution for regression testing and how to develop such a framework. An essential aspect of this thesis is the emphasis on the broad domain knowledge required for crafting effective test cases in a domain-specific context. This knowledge is crucial for understanding the intricacies of the domain, which directly influences both what to test and how to structure the test cases.

The following chapters discuss the background of the problem, our problem formulation and motivation, how we managed to develop a test framework, and an analysis of the automation process.

This thesis aims to contribute to the knowledge about how one can develop an automated regression testing solution, along with an applicable automation process for companies facing similar problems, as they operate towards a convenient and time-saving process for conducting regression testing.

While the solution presented in this thesis might initially appear akin to general test automation, it is important to underscore that the solution’s scope and primary objective are distinctly aligned with regression testing. The developed solution is specifically designed to automate a predetermined set of test cases as part of each deployment cycle. With the goal of ensuring that every new software deployment is rigorously evaluated for potential regressions. While the solutions characteristic relate to broader test automation, the solution’s core is rooted in the principles and objectives of regression testing, distinguishing it from more generic test automation frameworks.
2. Background

The background section of this thesis discusses fundamental concepts in the field of software testing, specifically as they relate to the domain of test automation, Software-as-a-Service (SaaS), and regression testing. The section defines SaaS and its typical characteristics. It explores the concept of regression testing and its role in software maintenance, emphasizing the need for thorough testing of both individual components and the system as a whole. Test automation is also introduced as a means of increasing the efficiency and effectiveness of regression testing. Still, it acknowledges the challenges and costs of developing and maintaining automated testing tools. The section also briefly discusses the challenges domain-specific SaaS providers face when finding appropriate third-party vendors to develop automated testing solutions. Overall, it creates a foundation of knowledge to further investigate regression testing automation for domain-specific SaaS systems.

2.1 Essential Background and Terminology

In the following sections, we continue to explore some of the fundamental concepts needed for this paper and the challenges of automating regression testing, and the potential benefits of automation. By examining these concepts in more detail, we aim to better understand how they relate to our research questions and justify our proposed approach.

2.1.1 Software as a Service (SaaS)

Software-as-a-Service is a software deployment model that is hosted, provisioned, and accessed by the user through the Internet. There are different SaaS business models and Vendors sometimes disagree on what distances a true SaaS from a regular hosted software application. Nonetheless, the specified characteristics are considered common practices that apply to all SaaS models [4].

- **Web Access:** Users access the software application through the web using their preferred browser.

- **Subscription pricing:** Most often paid as a recurring monthly fee instead of a one-time upfront license.

- **Updated:** Functionality is controlled and updated by the vendor, with enhancements and fixes being patched multiple times per year/month.

- **Low customization:** Barely any customization options as the application is standardized across multiple customers and often hosted in a multi-tenant architecture model.

2.1.2 Regression Testing

Regression testing is a crucial aspect of software maintenance and is a testing process executed after a software program or system update. Its purpose is to ensure that the performed modifications do not introduce any unintended behavior in both the affected components and the overall system. Therefore, this process requires both testing of the impacted sections of the software program and the system as a whole [5].

2.1.3 Test automation

Test automation of the process of automating repetitive software testing, most often with the help of a specialized software tool or framework [6]. Test automation can help speed up the testing process, reduce its costs and increase its effectiveness.

2.1.4 Domain Specific SaaS

Domain-specific systems are systems or software specifically designed to meet the unique requirements of a particular domain. Domain-specific systems are prone to complex solutions and may require special tools and techniques for testing [7].
2.2 Regression Testing in Domain-Specific SaaS

Testing is an essential aspect of any software development cycle, and different companies are experiencing several challenges [8], [9], especially when the software system is large and complex. One of the most demanding testing activities is regression testing, which can extend to up to 80% of the combined cost for testing. Several articles recommend developing an in-house regression testing solution in SaaS settings due to the inability to find fitting third-party vendors [8]–[10].

2.3 Exploring the Benefits and Limitations of Test Automation

Test automation is usually done with the help of software or a tool. A common practice is to either develop an in-house built test tool or purchase one from a third-party vendor [8].

Test automation has been shown to offer several benefits; when used with tests designed with maintenance in mind, it supports both re-usability and repeatability. These benefits are particularly relevant for regression testing in domain-specific settings where several rounds of testing may be required. The possibility of repetitiveness advocates confidence and less human effort for testing, presenting more time for other activities such as error prevention. Moreover, since test automation also is shown to enhance test coverage [11], it is not only beneficial when frequent regression testing is needed.

Although test automation is often associated with numerous advantages, it has limitations. Many of these limitations are attributed to cost factors, particularly the high initial expenses involved in purchasing or developing an automation tool[11]. Furthermore, automation initially requires a significant amount of time to be given to the design phase, regardless of the approach chosen by the company to implement the automation process. It has also been observed that automated testing is generally less effective in detecting complex bugs compared to manual testing in some contexts.
3. Problem Formulation

Software development and its associated activities are crucial for maintaining a well-functioning system, demanding ongoing attention and improvement. This requirement becomes even more significant in complex domains with a growing customer base, where updates and enhancements occur frequently. Continuous development and testing can be expensive, leading many organizations to seek assistance from third-party vendors, mainly for testing purposes. Unfortunately, working with complex and domain-specific systems makes it difficult to turn to third-party vendors, as domain-specific software may require significant customization. And third-party vendors may not have the necessary competence and knowledge to understand its unique features and requirements [12], [13]. This can lead to sub-optimal test coverage, inaccurate results, and a situation where the cost exceeds its value. Additionally, third-party vendors may be unable to replicate the company’s environment and configurations, further reducing the accuracy and validity of testing results.

In discussions with a domain-specific SaaS provider in Sweden, the challenges faced by such companies in addressing regression testing issues were brought to light. This highlighted the complexities encountered when seeking solutions for regression testing challenges within the domain-specific context. Specifically, we discovered the continuous development of their system, which necessitates ongoing testing. While the company had explored the possibility of engaging third-party vendors to address their regression testing problems, they found that most vendors lacked the necessary domain experience and knowledge to develop an effective automated solution. This company was founded in 2013 and has since experienced rapid growth. The company is owned by three of Sweden’s largest insurance brokers and acts as a SaaS for its owners’ customers. One of the company’s first and now largest systems currently serves over 20,000 companies. Such a large clientele brings multiple differences, where some were not considered when creating the system. This has now led to increasing pressure on testing to ensure the reliability and security of the software. The company under study is facing known challenges with regression testing [7], where they are struggling to complete their regression testing in the time frame and therefore, cannot guarantee sufficient test coverage.

Originally, a team of manual testers managed all the regression tests, but with the system’s rapid growth and original design, complete test coverage is no longer manually manageable. This leads to uncertainty and test personnel wishing that the selected test cases were correct. Since domain specificity makes third-party vendors inappropriate, this thesis will investigate how to develop an automated solution for regression testing domain-specific systems to improve test coverage and reduce test time. Thus, the central research questions of this thesis are:

- **RQ 1**: What challenges are faced when implementing an automated solution for regression testing in a SaaS setting?
- **RQ 2**: What is the feasibility of automating regression testing for SaaS domain-specific systems?
- **RQ 3**: How can one develop a test automation solution to address regression testing challenges in domain-specific SaaS systems?

By answering these research questions, we aim to provide a practical and effective solution for improving regression testing in domain-specific systems, which can be implemented by the company in question, and potentially benefit other companies facing similar challenges.
4. Research Methodology

The research methodology used includes qualitative methods since it focuses on solving a specific problem for a particular group of systems. This approach tests the problem-solving approach on one system within a single domain.

The process found in this thesis is based on the three RQs identified in Chapter 3., and the method is a modified version of the state-of-the-art model for technology transferring made by Gorschek et al. shown in Figure 1. The model created by Gorschek is well suited for breaking in sub-activities in technology transfer, with each part responsible for providing sufficient information needed to solve each problem [3]. The model is modified with the goal of answering the research questions and creating an acceptable solution for the company under study.

![Figure 1: Original Technology Transfer Model](image)

The model is based on gathering both generic information and company-specific issues. The original model is divided into seven parts, some of which are distinct, making it feasible to deconstruct the work required for the problems into the models’ different segments. We have chosen to keep the design of multiple distinct parts for the modified model, which is displayed in Table 1.

Initially, we thought of using the original model shown in Figure 1, but after considering the scope of the thesis, we decided to modify it and create our own minimized version (Figure 2), only containing the parts necessary for the execution of this thesis shown in Table 1.
Step 1: Given a problem from the company, identify industry need and specific company issues related to the problem. Observe and interview to find issues.

Step 2: Using several resources and sources to find topics for research and formulate problem statements while studying the field of the thesis.

Step 3: Formulate a candidate solution with the aim to answer the formulated problems in Step 2.

Step 4: With the findings from 1 and 2, discuss and evaluate the solution in 3.

Step 5: Continue discussing the complete automation process, consider ethical standpoints and further development.

Figure 2: The modified technology transfer that aligns with the specific need of this thesis

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Problem identification, domain issues</td>
<td>We use interviews to identify company-specific problems and literature reviews to identify problems related to regression testing, test automation, and SaaS testing.</td>
</tr>
<tr>
<td>2.</td>
<td>Problem formulation</td>
<td>The results from the data collection is thematically analyzed to define the problems in a clear and concise manner and organize the identified problems into themes.</td>
</tr>
<tr>
<td>3.</td>
<td>Solution development</td>
<td>With the themes in mind, we create a draft solution on how we would try to automate regression testing in a domain-specific SaaS setting.</td>
</tr>
<tr>
<td>4.</td>
<td>Solution review, discussion</td>
<td>Evaluating, and discussing the proposed solutions to determine their effectiveness and feasibility.</td>
</tr>
<tr>
<td>5.</td>
<td>Process analysis</td>
<td>Assessing and examining the overall process to identify areas for improvement and optimization, along with discussing the possible ethical consideration of automation.</td>
</tr>
</tbody>
</table>

Table 1: Steps in the problem-solving process
4.1 Data Collection

Following the method, the first step towards answering RQ1 includes problem identification and investigating possible issues. Two standard methods for lessening a knowledge gap are interviews and literature reviews, both suitable for different situations. Interviews are great for gathering qualitative information, while it is more convenient to gather large amounts of data by studying existing literature.

4.1.1 Literature Review

The researchers will thoroughly review relevant with the goal of identifying existing knowledge and gaps in the current understanding of any of the following topics: domain specificity, regression testing, and test automation. Literature reviews will also be used to study the state-of-the-art ways of tackling common topic-specific issues and the domain-specific issues identified during the interviews. The literature for this review was primarily obtained from reputable academic sources. Google Scholar was utilized as a search engine to identify relevant academic articles and papers. The most eminent databases providing relevant literature for this thesis where: 1 IEEE Xplore Digital Library, 2 ACM Digital Library, and 3 Diva (Digitala Vetenskapliga Arkivet).

These databases provide access to academic publications in various fields, including computer science, engineering, and related disciplines. By utilizing these resources, a comprehensive collection of relevant literature was compiled for this review. It is worth noting that the search terms and filters used within each database were tailored to the specific research topic, ensuring the retrieval of relevant articles and papers. Common search terms were (regression testing, test automation, SaaS, automate regression tests in SaaS).

4.1.2 Interviews

The chosen approach to address the lack of domain knowledge and identify system-specific issues is through the use of interviews [14].

Interviews align with this thesis’ qualitative aspects, and this step increases the understanding of the domain and system. Interviews are a common form of data collection, where two of the most common approaches to interviews are structured and semi-structured. The difference is how the interviews are held and structured, where the latter being more flexible than the first. The interviews are expected to help develop a comprehensive understanding of the system and domain in question and a complete understanding of the current testing flaws [14].

4.2 Interview Design

The first step when designing an interview scheme is to define the research objectives, describe the problem, and present a scope of context for the RQs. According to the guidelines presented by Linåker et al., [15], the objectives of the design process of an interview should include (result usage, motivation explanation, resources needed to complete the interview). To maximize the benefit of the interview, it is also essential to consider the interviewees and their characteristics; what are their relevant experience, domain knowledge, and technical abilities? Further, it is necessary to consider the number of participants when designing the interview-scheme [15].

The following concepts must be considered fundamental when designing an interview. It is essential to consider various factors, such as defining the measurement criteria, aligning interview questions with the research questions, selecting the appropriate question type, establishing an effective question sequence, avoiding biased questions, and carefully choosing participants to ensure the validity of the interview results [15].

With these concepts in mind, along with the guidelines provided by Strandberg [14], the following is the interview scheme created for this research. Note that the interviewees are all Swedish; thus, the interview is done in Swedish. The scheme shown in table 2 is a translated version.

---

1 https://ieeexplore.ieee.org/Xplore/home.jsp
2 https://dl.acm.org/
<table>
<thead>
<tr>
<th>Section</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before we start</td>
<td>1. Motivation</td>
</tr>
<tr>
<td></td>
<td>2. Research goal</td>
</tr>
<tr>
<td></td>
<td>• What’s your role?</td>
</tr>
<tr>
<td></td>
<td>• What’s your expertise?</td>
</tr>
<tr>
<td></td>
<td>• How many years of experience do you have in the following topics &amp; how</td>
</tr>
<tr>
<td></td>
<td>would you estimate your competence 1-5.</td>
</tr>
<tr>
<td></td>
<td>– Test automation</td>
</tr>
<tr>
<td></td>
<td>– Regression testing</td>
</tr>
<tr>
<td></td>
<td>– Domain knowledge</td>
</tr>
<tr>
<td></td>
<td>2. Part 2.</td>
</tr>
<tr>
<td></td>
<td>• What would you consider the positive gains with an automated regression</td>
</tr>
<tr>
<td></td>
<td>testing solution?</td>
</tr>
<tr>
<td></td>
<td>• What are the biggest potential issues and/or potholes to avoid with</td>
</tr>
<tr>
<td></td>
<td>the development of an automated regression testing solution?</td>
</tr>
<tr>
<td></td>
<td>• What are the current flaws of your routine for regression testing?</td>
</tr>
</tbody>
</table>

Table 2: Interview design

To identify more specific issues, the interviews contained open answers and led questions designed to catch eventual "common issues" noted during the literature review. The interviews were conducted with both test personnel and other domain-competent colleagues. The presented scheme is in line with the goals of the interview, and despite being considered a structured interview, it gives the interviewee room for honest answers.

4.3 Data Analysis

To enhance the utility of the gathered data, we have chosen to conduct a thematic analysis of the combined literature and interview results. Thematic analysis is a qualitative research method used to identify, analyze, and interpret patterns, themes, and meanings within a data set. It involves systematically organizing and categorizing the data to uncover recurring ideas, concepts, or patterns of significance. Researchers engage in a rigorous process of coding, categorizing, and grouping the data to generate meaningful themes or patterns that capture the essence of the data. The thematic analysis allows for a deep exploration of the data, enabling researchers to gain insights into the underlying themes, variations, and connections within the data set. It is a flexible and widely used approach in qualitative research, allowing for exploring diverse research questions and identifying rich and nuanced findings.
5. Interview & Literature Review Results

This section will display recognized themes and identified patterns found while conducting the thematic analysis.

5.1 Interview Results

In Table 3 and Table 4, we present the experience of each interviewee, their crucial success factors, possible issues that may hinder development, and the primary point of improvement to consider during the development of the test solution. The company under study had previously identified a testing problem in their most extensive system, where the manual testing approach that was once sufficient no longer provided efficient time management, particularly given the current frequent release pattern and the complexities within the domain.

One notable concern raised during the interviews is the risk of automation bias in the context of testing. As one interviewee mentioned, "With an automated solution, there is the risk of automation bias, where we might tend to rely overly on automated tests due to complacency." This highlights a potential challenge where there could be an overreliance on automated tests, assuming they are infallible and neglecting the importance of manual testing in certain scenarios.

In conclusion, while automated testing offers significant advantages in terms of handling the large matrix of tests that are now too complex for manual testing, it is important to remain vigilant about the potential for automation bias. The SUT handles various input variants, ensuing complex test cases. Together with the high workload, it makes the current testing process prone to human errors, resulting in false test results.

<table>
<thead>
<tr>
<th>Interviewee</th>
<th>Role &amp; Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interviewee 1</td>
<td>Role: Tester, Test automation: &lt;1 year, Regression testing: &lt;1 year, Domain knowledge: 10 years</td>
</tr>
<tr>
<td>Interviewee 2</td>
<td>Role: Tester, Test automation: &lt;1 year, Regression testing: &lt;2 year, Domain knowledge: 2 years</td>
</tr>
<tr>
<td>Interviewee 3</td>
<td>Role: Tester, Test automation: &lt;1 year, Regression testing: &lt;4 year, Domain knowledge: 8 years</td>
</tr>
</tbody>
</table>

Table 3: Interviewees’ experience
Table 4: Success factors and challenges in test automation

<table>
<thead>
<tr>
<th>Interviewee</th>
<th>Success factors of an automated test framework</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An automated test framework would allocate additional time for other testing activities. If we are able to speed up the execution of regression testing, we will likely achieve comprehensive test coverage.</td>
<td>With an automated solution, there is the risk of automation bias, where we might tend to rely overly on automated tests due to complacency. It is also important to consider the domain competence of those in charge of establishing the test cases. If the test cases are inaccurate, so will the result.</td>
</tr>
<tr>
<td>2</td>
<td>With an automated test framework that speeds up the execution of repetitive tests, we could allocate more time for other testing activities. As the current workload is too high to achieve full test coverage, there is a tendency to stress through some repetitive tasks. This could be avoided with an automated solution.</td>
<td>Possessing both domain and test-specific competence is crucial for creating reliable tests. Finding this may be challenging.</td>
</tr>
<tr>
<td>3</td>
<td>An automated solution would help us run repetitive tasks more frequently and prevent human errors that may occur when rushing through these repetitive tasks. Finally, it could reduce the cost of regression testing.</td>
<td>&quot;I have experienced a challenge in finding people who want to work with test development, my insight is that most developers tend to lean towards software development instead of test development.&quot; There is also a challenge in finding competence related to test automation, and test automation requires an equal amount of maintenance as manual testing.</td>
</tr>
</tbody>
</table>

5.2 Literature Review Results

While numerous studies have been conducted on test automation and regression testing [16]–[19], there is comparatively less research available specifically on automating regression testing for domain-specific SaaS systems.

Regression testing is performed to validate code changes and ensure they do not introduce unintended behaviors or defects. One approach to regression testing involves re-executing all existing valid test cases, which can be both time-consuming and costly [16].

Antonia Bertolino discusses the many faces of software testing [20] and points out the importance of identifying the why, how, how much, what, where, when when conducting software testing. In addition to acknowledging that testing involves observing a subset of executions and drawing conclusions based on them, she highlights various challenges encountered in software testing. She addresses the challenges related to domain-specific testing approaches when discussing test automation. Furthermore, she emphasizes the importance of incorporating domain expertise when developing test frameworks.

Avik Sinha and Carol Smidts discuss the test design for domain-specific applications and emphasize the crucial elements of automating black-box testing. They highlight that effective automation requires creating test cases based on the SUTs’ structure and behavior. They also emphasize the importance of domain knowledge in achieving successful automation [21].

Regression testing practices are discussed in a research paper by E. Engström and P. Runeson in 2010 [22]. The paper highlights that regression testing becomes increasingly crucial as software projects grow in complexity, with a greater emphasis on systematic reuse and shorter development
cycles. It delves into how regression testing should be conducted. Furthermore, the paper presents a qualitative survey investigating regression testing practices in various domains, aiming to generate knowledge about the testing industry and the research community. The survey covers topics such as the importance and challenges of test automation, specific challenges in regression testing, as well as general testing challenges [22].

A study by Garousi et al. [18] examines test automation and presents a checklist to assist in deciding whether to automate testing. The checklist encompasses factors grouped into five sections, including software-under-test-related factors, test-related factors, test-tool-related factors, human and organizational factors, cross-cutting, and other factors. These characteristics contribute to the automation process or may counteract its potential benefits. On the other hand, Mäntylä et al. discuss the benefits and limitations of automated software testing in their study [11]. The authors conducted a practitioner survey, where most participants agreed to the incompatibility of available testing tools on the market. They also acknowledged the initial cost associated with automated testing but expressed skepticism that automated tools would completely replace manual testing.

A research paper conducted at ABB highlights potential pitfalls to consider and avoid when considering and developing an automated test system. The paper emphasizes the importance of having a detailed test specification to reduce development time and minimize the need for changes resulting from specification revisions during development. Starting with a small-scale implementation rather than attempting to incorporate too many test cases at once is identified as one of the pitfalls to be aware of [19].

Kaspar van Dam explores the future of software testing in his article from 2019 [17]. He highlights the potential for an expanded role and increased responsibility for testers. Van Dam also references Darwin, citing a relevant quote: "It is not the strongest one that survives, nor the smartest. But the one most adaptable to change." Putting emphasis on these aspects to stay relevant in the profession of a software tester, one needs to learn test automation and its surrounding technologies.

A research paper by Abhinav Sharma et al. [23] discusses API testing and highlights the exponential increase of API calls between systems. Especially when distributed software applications interact with back-end utilities through standardised interfaces.

In Table 5, a concise summary of findings from several of the previous studies is presented. The studies collectively highlight key insights in the field of software testing and automation. One notable finding is the complexity of automating tests in domain-specific settings, which often require specialized tools. Regression testing emerges as a commonly automated process due to its repetitive nature. Challenges associated with automation include result visualization and process monitoring. Additionally, the results emphasize the importance of taking a cautious and prudent approach when automating tests, suggesting that automating the entire test suite at once may not be advisable. Ethical considerations are also brought into focus, urging practitioners to carefully assess the ethical implications of automation, particularly in the context of regression testing. Overall, the literature review results provide valuable insights into the domain-specificity of automation, challenges in SaaS testing, the benefits and challenges of regression testing automation, prudent automation practices, and ethical considerations in test automation.

5.3 Recognized Themes

During the interviews, several themes appeared (as shown in Table 6). Most importantly, all interviewees specified that the cost of performing regression testing is the main issue of today’s regression testing routine. This was also distinguished as the leading cause for examining the possibilities for an automated regression testing solution. The theme of the high cost related to regression testing repeats itself throughout the literature reviews, where it is prevalent noting that regression testing is the most time-consuming component of the typical testing pipeline.

Another theme that emerged from both the interviews and the literature review is that domain competence is mentioned as the main hindrance in finding compatible third-party vendors. The significance of domain competence becomes a recurring topic to consider when designing the process of developing an in-house solution for regression testing. It is, therefore, necessary to highlight the importance of domain competence, independent of whether the company considers developing an in-house testing solution or examining the potential of buying one from a third-party vendor.
Oskar Renefalk  

Automating Regression Testing

Citation Coverage Findings

[18], [20] Domain-specificity, Regression Testing, & Test Automation  
Automation gets complicated in a domain-specific setting and often requires specially developed tools. Regression testing is the most sought-after testing principle to automate.

[9], [10] SaaS Testing and Automation  
SaaS testing and test automation are often done by the customer, proving problematic. An in-house developed tool is recommended.

[16], [22] Regression testing  
Regression testing is often automated due to its repetitive nature. Identified challenges are result visualization and process monitoring.

[19] Pitfalls of Automation  
Avoid the temptation of automating the whole test suite at once. Take a prudent approach.

Test automation is the future, but be cautious and consider the ethics of automation when endeavoring an automated solution for regression testing.

Table 5: Summary of the findings from previous studies

Nonetheless, it is worth noting that the test manager highlighted "finding the right people" as a potential pitfall in the automation process. This emphasizes the importance of competence not only within the specific domain of the company but also within the domain of software testing.

During the interviews, it was established that the test suite has become significantly extensive for a manual process, making it challenging to comprehend every test case’s complexity fully. The testers mention the "prudent approach" as one of the critical factors for a successful automation process, which is backed in the research paper by Christer et al. [19], where the authors talk about the causes of a failed automation process and mention the urge of trying to implement all, or many, test cases at once as one of the significant pitfalls to avoid. To avoid this, we decided to limit the test framework’s scope as much as possible.
<table>
<thead>
<tr>
<th>Theme</th>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost Reduction Benefits</td>
<td>1.</td>
<td>All interviewees mentioned time as the main issue of today’s testing and the main profit from an automated solution.</td>
</tr>
<tr>
<td>Domain-Competence</td>
<td>2.</td>
<td>Both interviewees and literature reviews point to domain competence as crucial when creating an automated test framework.</td>
</tr>
<tr>
<td>Prudent Approach to Test Automation</td>
<td>3.</td>
<td>Both the literature reviews and the input from the interviewees strongly emphasize the view that attempting to automate all test cases is an ineffective strategy destined to fail. This main theme underscores the importance of adopting a more thoughtful and measured approach to test automation, focusing on starting small rather than trying to automate everything at once.</td>
</tr>
<tr>
<td>Balancing Test Competence with Domain Competence</td>
<td>4.</td>
<td>This emphasizes the need to strike a balance between a deep understanding of the testing process and possessing the necessary knowledge and expertise related to the particular domain.</td>
</tr>
</tbody>
</table>

Table 6: Main themes of the interview

6. Development of the Test Automation Framework

This section of the thesis covers the design process and development of a test automation framework. The development phase is designed and carried out based on the themes identified during the thematic analysis in Table 6, as well as the initial specific company requirements.

This framework, representing the initial iteration of a framework intended for future development, has been designed with a focused scope deliberately minimized to serve as a practical starting point and mitigate potential pitfalls. It will offer a comprehensive overview of the system structure and address the associated limitations through detailed descriptions and discussions. The architecture is created based on the insights gathered during the interviews and the literature review results, which were analyzed to recognize key themes. The themes serve as a major foundation for the development phase, affecting every aspect of the framework’s design. Moreover, this section explores the relationship and communication between the test automation framework and the SUT, and how its inputs affect the scope and specifics of the testing framework.

Furthermore, building upon the identified themes, we delve into the test execution process, result file creation, and result validation. To reinforce the credibility of our development process, we incorporate examples that demonstrate how the test cases were designed, considering the insights gained from the pothole prevention suggestions [19]. This section ends with a discussion of the SUT limitations and why the SUT does not support a fully automatic test framework. Together, this section covers the complete development process and gives a comprehensive understanding of the test automation framework, its intricacies, design choices, and potential ideas for further development.

6.1 System Architecture

Our goal was to create a design that effectively caters to the unique requirements of the System Under Test (SUT). As highlighted in Table 5, both domain-specificity and SaaS testing necessitate distinct approaches to automate regression testing [20]. Therefore, it is crucial to identify and understand these unique requirements before proceeding with the development of an automated solution. To achieve this, we conducted a thorough literature review and conducted direct interviews, which provided a significant amount of data for analysis. Through this data collection approach, we identified several key themes that directly influenced the structure and functionalities of our framework.
We begin by giving a conceptual overview of the system architecture before providing further information about the connectivity between the framework and the system.

Figure 3: Test Automation Architecture Overview

In Figure 3, we present a brief overview of the designed workflow for the testing solution. This section covers these steps shortly, ahead of further in-depth exploration:

• 1: The tester starts by selecting a stored test case represented in JSON format containing the XML file used for input in the SUT, along with a result file containing the expected result of an executed test.
• 2: The input file specific to the SUT is extracted from the test case.
• 3: The extracted XML file is transmitted as input to the SUT, while the test case is provided as input to the test framework.
• 4: The data exchange phase commences, and the test framework retrieves data from and supplies data to the SUT. The retrieved data is utilized to generate a result file.
• 5: The test results are displayed visually, enabling the tester to inspect them.

6.2 SUT Limitations and Recommendations

Considering future maintainability, the decision has been made to develop the test framework using the same technology stack employed by the concerned company. This technology stack includes .NET C#, Python, and Angular.js. Aligning the test framework with the existing technology stack, facilitates seamless integration, easier maintenance, and ensures compatibility with the company’s established software ecosystem.

The SUT is a complex and large system, making it impractical for a detailed inspection. Therefore, this section aims to provide an overview of its key components and their impact on the testing framework rather than conducting a comprehensive analysis of the entire SUT. The SUT, a .Net application written in C#, serves as a SaaS for roughly 20 000 companies in Sweden. This entails a need for adaptability which in turn requires continuous releases.

In production, the SUT takes an XML file input, the format of the file is different depending on where it is created. There are two versions, one created by another in-house application and one created by a specific insurance company. The SUT utilizes some variances in behavior depending
on the input type. The SUT exhibits several unique characteristics, one of which pertains to data exchange. Notably, the data exchange process in the production environment differs from that in the testing environment. In production, all data exchange is controlled by a secure web service called SSEK \(^4\). SSEK is a Swedish service that ensures that sent data will get through and that expected data is received. In the testing environments, this is mocked with an in-house developed mock application. The SUT sends data in the same way as in the production environment, but instead of communicating with the SSEK, it communicates with the mocker. The data exchange is carried out through the use of API. During the first development iteration of the test framework, we take advantage of the mocker application and its existing configuration and use it to speed up the development of the test framework.

The SUT is controlled by an ASP.net webforms GUI and utilizes an in-house developed task manager, batching work and executing several tasks in parallel. Due to an insufficient task manager, the current version of the SUT lacks support for a fully automated testing solution. The existing task manager lacks a way to notify the test framework when a set of tasks is completed, resulting in the test framework being unaware of when to retrieve data from the SUT.

6.3 Test Case Model

The input file decides the SUT’s behavior and the expected output. This leads to the test case model being characterized by the in-production input files. What distinguishes the test file from the actual input file is the absence of real personal identity numbers and an actual company registration number.

The test file is, as stated before, an XML file called a broker file designed explicitly for the SUT. There are two types of broker files, and both require different behavior of the SUT. There are also certain tags in the broker file that alters the behavior of the SUT. Before importing the broker file, the user should select the type. To avoid the identified pothole of trying to implement all cases at once \([19]\), and recognize Theme 3 in Table 6. Emphasizing the significance of a gradual approach, it is essential to start small and resist the temptation to implement an automated solution for all test cases simultaneously. We have opted to implement a test file with minimal complexity and reduced scope. The test case will be simplified by incorporating only a small set of variables that interact consistently with the SUT. While these variables are easily distinguishable, they require substantial domain experience to ensure accurate implementation.

6.4 Test Framework Design

The concept behind the test framework is to enable the parallel execution of multiple test cases, each containing numerous tests. These test cases consist of a combination of an XML file serving as input for the SUT and a JSON file that encompasses the expected result of a successful test framework execution. Together, this combination forms the input for the test framework.

Based on the themes identified during the thematic analysis, the test framework is assigned a specific set of tasks to accomplish as follows:

1. Generate input for the SUT.
2. Fetch correct data from the SUT (through the mocker application).
3. Set up a result file from the fetched data.
4. Visualise the result for the user.

Upon completing these tasks, the framework will perform all the essential steps involved in testing the System Under Test (SUT). To ensure comprehensive coverage, each of the four tasks will be further divided into sub-tasks, which will be tailored to address the specific idiosyncrasies of the SUT and build upon the previously developed Mocker application.

6.4.1 Input Handling

The input file for the test framework is in JSON format. This JSON file comprises two components: a string representation of the XML file used for invoice creation and as input for the SUT and a string representing the JSON file containing the expected result. The test framework saves the input by extracting the broker file and the result file, creating a test case shown in Listing 2, which is added to an SQL database. While this thesis narrows its scope to two brokers, the need to further support for additional XML schemes is redundant. However, to support the addition of new brokers, the test framework would require additional parsers to manage the new XML schemes.

6.4.2 Result File Creation

The result file is generated to provide a clear visualization of potential differences. We also needed to consider the specifics of the domain. Given that we are dealing with insurance information, it is crucial to include and verify key properties such as insurance numbers, insurance amounts, insured person details, and other relevant information. These properties play a vital role in ensuring the accuracy and completeness of the testing process. To prevent the potential consequences outlined in Theme 3 (shown Table 6), we have taken a cautious approach in selecting which functionalities of the SUT to include in the initial iteration of our test framework. We selected four functionalities: calculating salary changes, insurance premium calculation, payment specification creation, and customer invoices creation. The output from each executed functionality of the SUT contains insurance-specific data in some pre-structured data format, always as an XML. To create these files, the SUT utilizes C# classes made from XSDs, the company provided these XSDs in the interest of creating the test framework. This made deserializing the XML file into its suitable C# class possible. When iterating, comparing, and/or operating on data, working with the C# objects is more convenient than the string. The result file is constructed using the data fetched from each of the four parts. Deserializing it to C# objects and adding the needed data to the matching personal identity number. When all the parts have been added, a complete result file has been created and is now ready for validation.

Algorithm 1  Add desired data from executed SUT process to result file

<table>
<thead>
<tr>
<th>Require: fetched data</th>
</tr>
</thead>
<tbody>
<tr>
<td>function GetInsuranceNumbersFromBrokerfile</td>
</tr>
<tr>
<td>insuranceNumbers ← all insurance numbers from brokerfile</td>
</tr>
<tr>
<td>return insuranceNumbers</td>
</tr>
<tr>
<td>end function</td>
</tr>
<tr>
<td>function ExtractNeededData(fetchedData)</td>
</tr>
<tr>
<td>neededData ← needed data from fetchedData</td>
</tr>
<tr>
<td>return neededData</td>
</tr>
<tr>
<td>end function</td>
</tr>
<tr>
<td>fetchedData ← data fetched from SUT</td>
</tr>
</tbody>
</table>

function AddDataToResFile(fetchedData)
| insuranceNumbers ← GetInsuranceNumbersFromBrokerfile |
| for each fd in fetchedData do |
| if insuranceNumbers.contains(fd.insuranceNumber) then |
| dataToAdd = ExtractNeededData(fd) |
| resultFile.add(dataToAdd) |
| end if |
| end for |
| end function |
In Algorithm 1, we present a high-level pseudo-code representation of the code utilized to add data to the result file. When retrieving the fetched data from the mocker application, the function called `AddDataToResFile` (line 13) begins with retrieving all of the insurance numbers from the concerned broker file (line 14). The function continues by iterating through all the fetched data (line 15) to search for any data where the insurance number exists in the broker file. If such data exists, the function `ExtractNeededData` (line 6) extracts all the needed properties from the fetched data and returns. Back in `AddDataToResFile`, on line 18 we add the extracted data to the result file, preparing the result file for validation.

Since the XML files sent as outputs from the SUT are of three different types, the `ExtractNeededData` function in Algorithm 1 will vary depending on the specific type of XML file being fetched. These XML files may have slight differences due to being generated from different XSDs with corresponding C# type classes. However, the core functionality of the algorithm remains unchanged.

### 6.4.3 Result Validation

The result file and the expected result file are of the same structure. It is convenient to compare their C# structure, and from there log each difference. The file representing the expected result is a pre-made representation of what the testing framework should generate from the data given from the SUT. If there are differences between them, those should be visualized to the tester for further inspection. The result validation is done with the use of object comparison. A C# method that recursively iterates through both objects’ properties and compares them, returning a list of all found differences.

---

**Algorithm 2** High-level version of the code for comparing C# objects

1: procedure `ValidateProperties(name, testCaseId, obj1, obj2)`
2:     `changesLst ← new List<DifferenceModel>()`
3:     if either `obj1` or `obj2` is null then
4:         Add difference to `changesLst` if only one object is null
5:     return `changesLst`
6: end if
7: Determine type of `obj1`
8: if `obj1` and `obj2` are arrays then
9:     for each element pair in `obj1` and `obj2` do
10:        if element pair is unequal then
11:            Add difference to `changesLst`
12:        end if
13:     Recursive call to `ValidateProperties` for each element pair
14: end for
15: return `changesLst`
16: end if
17: if `obj1` and `obj2` are non-string class objects then
18:     for each property in `obj1` and `obj2` do
19:         Add differences of property values to `changesLst`
20:     Recursive call to `ValidateProperties` for each property
21: end for
22: return `changesLst`
23: end if
24: if `obj1` and `obj2` are unequal then
25:     Add difference to `changesLst`
26: end if
27: return `changesLst`
28: end procedure
Algorithm 2, represents a high-level pseudocode for the test frameworks functionality for comparing and validating objects. The function is used to search for differences between the result and the expected result. The algorithm begins by creating a new list of the type DifferenceModel, shown in Listing 1. The algorithm can be described by dividing it into the following four different parts:

- **Part 1 (Line 3-6) check for null**: These lines check if either obj1 or obj2 is null. If one of them is null and the other is not, it adds a difference to the difference list. But only if only one of them is null (since if both are null, they are actually identical). It then returns the list of differences changesLst and exits the procedure.

- **Part 2 (Line 8-16) check for null**: Handles the case where both obj1 and obj2 are arrays. It iterates over each pair of elements in obj1 and obj2. If a pair of elements are unequal, it is added as a difference to changesLst. Additionally, for each pair of elements, the procedure makes a recursive call to itself, allowing it to handle nested arrays. After checking all element pairs, it returns changesLst.

- **Part 3 (Line 17-23) Non-string class comparison**: Since strings are recognized as an object of the class String in C#, this part handles objects of non-string classes. Iterating over each pair’s properties. If the value of a property pair is different, it is added as a difference to changesLst. Like part 2, the algorithm recursively calls itself for each property, supporting validation for nested properties. After all pairs are checked, it returns changesLst.

- **Part 4 (Lines 24-27): Handling All Other Cases**: This generally includes simple data types like integers, floats, strings, etc., where obj1 and obj2 are neither null, arrays, nor non-string class objects. If obj1 and obj2 are unequal, a difference is recorded in changesLst. Finally, it returns changesLst.

In all parts of the algorithm, changesLst is used to keep a record of the differences found between obj1 and obj2. The algorithm is recursive, meaning it calls itself to handle nested structures like arrays and non-string class objects.

```csharp
public class DifferenceModel
{
    public int TestCaseId { get; set; }
    public string ExpectedResult { get; set; }
    public string Result { get; set; }
    public string AsString { get; set; }
}
```

Listing 1: DifferenceModel class represents a Difference, each Difference will contain the path of the difference, the expected result, the actual result, and a string explaining the difference in words

### 6.5 Test Execution

The need for visualization of test results was mentioned in the interviews, and the initial thought was to create a customized GUI using Angular.js. During the development of the test framework, we noticed that the time window needed to create a specialized GUI was too small, and we had to consider another approach. The already in-use mocker application is controlled using an Postman\(^5\), an API platform used for designing, developing, and testing APIs. Due to the pre-existing competence on how to set up and use Postman, we realized that it is the most time-efficient and convenient approach.

\(^5\)https://www.postman.com/
6.5.1 Postman

With Postman, the user can create different environments, making it convenient to change from, for example, DEV1 to localhost. It also lets the user create a collection of API calls, ideal for setting up test suites. These collections are shareable so that other team members can import them and get an identical copy of your collection. With the company’s current technology stack, most applications are ASP.NET applications with an Angular front end, executing back-end functionality through the use of API calls. Having an already set-up postman configuration, providing similar functionality as an Angular GUI, there is no motivation for developing an Angular GUI within the scope of this thesis.

6.5.2 Test Framework - Order of events

With the Postman software being set up and used as a GUI. The execution of the Test framework is straightforward. All of the following HTTP requests mentioned are being carried out with the use of Postman. The mocker application and the test framework perform their database access against an SQL database with the use of entity framework. In Listing 2 you can see the C# class used.

```csharp
public class TestCase
{
    [Key]
    public int Id { get; set; }

    public string BrokerFile { get; set; }

    public string ExpectedResult { get; set; }

    public string Result { get; set; }

    public IEnumerable<Invoice> Invoices { get; set; }

    public DateTime Date { get; set; }
}
```

Listing 2:_TestCase class represents a test case, each test case will contain an ID, a broker file, an expected result, and the invoices that are created for the specific test case.

Figure 4 shows the data exchange process between the test framework, mocker application, and SUT. As the test framework will use APIs for data exchange, the following HTTP requests are enumerated to match the steps in Figure 4, and are responsible for executing the test framework. The HTTP requests need to be executed sequentially as follows.

1. **HTTP post** - Upload test file from disc, TF processes the test file and creates an instance of the C# object testCase shown in Listing 2. It proceeds to upload the broker file and expected result to an SQL database with a GUID as the primary key.

2. **HTTP post** - Get all test cases from the DB, use the broker file to create invoices and update the DB with the created invoices, and also upload them to the mocked SSEK queue.

3. **HTTP post** - Get all outputs from the mocked SSEK queue, filter only to get salary changes. For each test case in the DB, update the result with the correct information from the messages.

4. **HTTP post** - Same as the previous Post, but now get messages containing insurance premiums.
5. **HTTP post** - Same as the previous Post, but now get messages containing payment specification.

6. **HTTP post** - Same as the previous Post, but now get messages containing customer invoice.

7. **HTTP get** - Validate result, get both the expected result and the result for each test case in the DB, verify the actual result, and return the created result verification file with all the possible differences. If there are no differences, return that.

After executing a test case, the framework returns a list of differences in a JSON format. Figure 5 displays the result of a test case where the insurance number and yearly salary differ. The expected salary was 500000, the reported salary was 700000, and the expected insurance number was 100055, but the reported was 10066.

![JSON file containing all reported differences from a specific test case](image)

Figure 5: JSON file containing all reported differences from a specific test case
7. Discussion

This section aims to clarify and forward the result of the created framework by thoroughly reviewing the complete automation process. Further, it discusses the reviewed solution and points out potential validity threats.

7.1 Test Framework Development Discussion

While the developed framework offers several benefits that significantly enhance the efficiency of the testing process, it is important to note that it is currently incapable of fully automatic execution. This limitation arises from the SUT’s inability to signal the completion of executed tasks.

Our initial aim was to compare the execution time of the current semi-automatic framework with a fully automated one. However, given the framework’s inability to function autonomously, a direct comparison of execution times may not provide a comprehensive picture. It is important to remember that the full advantages of automation, explicitly concerning execution time, can only be fully realized once the framework supports full automation.

Despite the abovementioned limitations, our framework supports other time-saving features that significantly streamline the testing process. One key feature is automated invoice creation. This function automates a time-consuming task that traditionally requires manual effort, speeding up the overall process.

In addition to this, the framework also includes a feature for test result visualization. This tool allows for a quick and intuitive understanding of the test results, saving time on manual data interpretation. Visual representation makes it easier to spot patterns, identify outliers, and make sense of complex data.

We integrated the tool Postman into the framework, which we utilized as an API runner and graphical user interface (GUI). Postman served to streamline the process of running and testing APIs, thereby contributing to the efficiency and usability of our testing framework.

It is important to note, however, that our initial approach was to develop a customized GUI specifically tailored to the testers’ requirements. However, due to time and resource constraints within the scope of this thesis, we had to seek an alternative solution. Fortunately, the mocker application had previously utilized Postman, significantly reducing the implementation period and the learning curve associated with using Postman. However, it is worth mentioning that Postman does not provide the same level of customization as a GUI developed from scratch, resulting in limited visualization of the test results, which we had to accept, given the project’s limited scope.

Overall, while Postman offered a satisfactory solution for the thesis, future work could focus on further enhancing this aspect to tailor the framework to specific testing needs and scenarios.

7.2 RQ1 Revisited - Identified Challenges

The primary objective of the thesis was to identify challenges encountered in a domain-specific SaaS setting while aiming for an automated regression testing solution. Through our research, we have identified several critical challenges and obstacles that must be addressed. One of the key findings is the importance of having domain-specific competence and expertise in testing practices. This leads to the second obstacle, which is finding and recruiting competent personnel. The test manager at the company in question highlighted the recruiting process for capable developers interested in test development as a significant concern. Apart from generic issues, the lack of support in the concerned company’s SUT for reporting the completion of tasks was identified as the primary challenge in automating their regression testing process. This limitation requires human intervention to perform certain tasks, preventing full automation.
7.3 RQ2 Revisited - Feasibility Of Automating Regression Testing For SaaS Domain-Specific Systems

In this thesis, we investigated the feasibility of developing and implementing an automated regression testing framework for a domain-specific SaaS. This exploration involved closely examining the development process, which served as the foundation for the testing framework created for the specific company under study. Despite the inherent complexities of domain-specific systems, our findings indicate that it is indeed technically feasible to create an automated testing solution for conducting regression testing in a domain-specific SaaS setting. Overcoming the posed challenges could bring considerable benefits in terms of increasing the efficiency and reliability of the automated testing process. Despite the potential hurdles and complexities, the positive implications of automation are compelling, thus making pursuing an automated testing solution a worthwhile endeavor.

7.4 RQ3 Revisited - Development Of a Test Automation Solution for Regression Testing

The third research question of this thesis is about how one can develop an automated solution for regression testing in a domain-specific SaaS setting. Our approach for answering this question was to implement a test framework for a company from Sweden within the insurance domain. While not yet fully automatic, the resulting framework presents a promising concept for future improvements, and the current test framework demonstrates the potential for developing a fully automated solution for regression testing in a domain-specific SaaS setting.

7.5 Validity Threats

There may be concerns about the reliability of the results presented in this thesis. With only three interviewees, all of whom are employees of the company being studied, there could be potential for biased or unrepresentative interview results due to the limited sample size. Nevertheless, having access to industrial engineers in this thesis is representative for this domain-specific setting. The test framework created as part of this thesis is specifically tailored to the SUT at the company in question, which might also influence the reliability and external validity of the thesis. It is conceivable that a different company or SUT could uncover distinct company-specific problems, leading to a differently structured framework and, consequently, different results. However, the developed framework and its architecture could be a starting point for companies with similar SUTs.

Furthermore, the thematic analysis and the development of the test framework were both carried out by a single individual. It is plausible that another researcher studying the same material might identify different themes or design a different framework, potentially leading to alternative findings. Nonetheless, the thematic analysis results were in line with the existing literature.
8. Ethical and Societal Effects of Test Automation

While an automated regression testing solution could revolutionize the testing process for any company deciding to implement it, it becomes crucial to delve into technological advancement’s ethical and societal impact. This section intends to examine the potential effects of such a solution. Some key aspects to explore are; privacy, bias, reliability, accountability, social impact, and human impact. Exploring these concepts can create a broader view of the automation process and its potential subsequent outcomes.

8.1 Privacy and Confidential Data

As the company encounters confidential data daily and has its management standards. Several precautions are pre-implemented, and the ethical consideration of privacy is managed.

Almost all the data used for testing purposes consists of example data that closely resembles production data without including actual personal information. The testing is today done on made-up companies with Skatteverkets made-up social security numbers, specifically created for testing.\footnote{https://www7.skatteverket.se/portal/apier-och-opppna-data/utvecklarportalen/oppetdata/Test%C2%AD%C2%ADpersonnummer}

8.2 Automation Bias and Accountability

Automation frequently introduces the phenomenon known as "automation bias" \cite{24}. This bias refers to the inclination to overly rely on the automated solution. When the system is reliable, test automation can enhance performance. However, in cases where the system has shortcomings not covered by the test suite, such as issues in the SUT code, these may remain unnoticed for multiple releases despite the tests indicating that the system is functioning correctly.

K. Van Dam discusses several aspects of accountability within a software tester’s role that may arise in the near future due to test automation \cite{17}. Particularly noteworthy is the aspect of "Accountability for quality assurance." As mentioned in Section 8.3, the role of software testers and their tasks are expected to shift focus in the coming decade, with the emphasis being placed more on ensuring the accuracy of predefined test cases and scripts for automatic tests rather than the actual execution. Consequently, the level of accountability a tester assumes may increase with automation rather than decrease.

8.3 Social and Human Impact

K. Van Dam highlights the current state of software testing as outdated. He draws attention to Charles Darwin’s well-known quote to illustrate the direction and perception of software testing: "It is not the strongest animal that survives, nor the smartest, but the one most adaptable to change" \cite{17}. Van Dam suggests that the rapid changes and adoption of automated testing are reshaping the definition and responsibilities of a "Software tester," and individuals who do not embrace the trend of automation may face the risk of job loss.

Although automation significantly improves the efficiency of test execution, it has the potential to impact the necessity of manual testing. While automation alleviates testers from manual testing tasks, such as repetitively executing regression tests, it may also pose a future challenge to the role of testers, requiring them to adapt and acquire the skills necessary for test automation. This shift highlights the importance of continuous learning and upskilling for testers to remain relevant in an evolving landscape. As mentioned in Section 8.2, the role of software testing is expected to expand in the next decade, with automated testing carrying increased responsibility for ensuring the reliability and correctness of tests and the scripts that execute them \cite{17}. This expanded scope of responsibility can lead to potential issues. Firstly, there may be an unfair distribution of workload among testers. Secondly, employees are required to learn new skills, and according to Van Dam, it should be the employer’s responsibility to provide adequate resources for proper education and training. Otherwise, the heightened demand for testers with modern abilities may lead to increased stress and potentially trigger imposter syndrome among employees \footnote{https://www.dictionary.com/browse/impostor-syndrome}.
It is important to consider how the expanding implementation of automated testing affects the testers used to today’s testing practices. In a survey from 2014, [25], researchers asked about regular testing practices, what makes a great tester, and how automated tools are used. They determined that knowledge about the specific testing technique, domain knowledge, dedication towards the craft, and an open mindset/willingness to learn new things makes a great software tester. This relates to the findings and foreseeing in the K. Van Dams paper [17], and further establishes the need for education on test automation. Otherwise, we might end up with an increased cleavage between the testers with access to the education needed to adapt, while those without adequate training could be left unemployed.

While test automation is an attractive solution and the projected future of software testing, it is essential to recognize and manage any ethical issues from the beginning. Companies that proactively consider the implications of test automation could ensure that the benefits of test automation do not come at the expense of job security, accountability, and societal equity [17], [25].
9. Conclusions

In this thesis, we have explored the possibility of automating regression testing in a domain-specific SaaS setting. Regression testing is considered the most time-consuming test practice, and due to its repetitive nature, it is also the most sought-after test practice to automate. Software is often automated with the help of external tools, usually purchased from a third-party vendor. However, third-party tools have been shown not to provide a sufficient solution when the software is highly-specific to a certain domain.

We identified the problem of automating regression testing in a domain-specific SaaS setting, correctly the Swedish pension and insurance industry, performing this thesis with the goal to do so.

The thesis research model is a modified version of Gorscheks information transfer model. The thesis follows the modified model through the entire automation process, which begins with problem identification. We identify and present challenges such as high cost, automation bias, frequent releases, and system complexity as issues related to regression testing, test automation, SaaS, and domain specificity. We performed a thematic analysis on the gathered information, creating several themes.

To answer the feasibility of automating regression testing in a domain-specific setting. We continued by developing a test framework using the identified themes from the thematic analysis as key concepts to follow.

As stated by [25], the goal should not be automation itself, but rather using automation as a means to achieve specific objectives. While adopting an automated approach is often beneficial for enhancing the regression testing process, there are certain challenges that must be addressed. In this thesis, we have presented the development of an initial test framework and automation process that demonstrates the feasibility of automating regression testing in a domain-specific SaaS setting. The developed framework presents a way of developing a test automation solution to address regression testing issues in a domain-specific SaaS setting. Since the thesis only studies one company, the test framework should not be considered a "one size fits all" solution but instead provide a way of working for domain-specific SaaS provides trying to automate their regression testing process. It is important to note that the architecture of the test framework is developed with regard to the SUT design and company desires, and companies exploring this must develop theirs as such. Therefore, this thesis should be used as a guideline for how to tackle the issue of automating regression testing in a SaaS setting, following the process of "problem identification - solution development - solution analysis". Our findings are supported by previous studies that have taken a similar approach to developing in-house testing solutions [9], [10].

9.1 Future Work

To further enhance the developed test framework, we need to ensure the SUTs’ support to announce the completion of each task. While it is possible to improve the test framework without the support for complete automation, it would likely provide the most prominent progress. As mentioned in section 7.1, Postman provided a sufficient GUI for the scope of the thesis, but further development should consider the development of a customized GUI to better visualize the results. A customized GUI could also introduce support for the tester to monitor the testing process and the completion of test cases.
References


