The Effectiveness of Implementing Automation System on Software Testing using Selenium WebDriver

Capstone Project

Submitted to the Department of Business Analytics

Sunway University

In Fulfillment of the requirement

For Postgraduate Studies

September 2024

DEDICATION

This research is dedicated with heartfelt gratitude to our families, whose unwavering support and encouragement have been our constant source of strength and motivation. Their belief in our abilities has driven us to overcome challenges and strive for excellence.

We also dedicate this work to our respected supervisor and lecturers, whose wisdom and guidance have been instrumental in shaping our academic journey. Their commitment to our growth has inspired us to pursue knowledge with curiosity and determination.

Lastly, we extend this dedication to our friends and classmates, whose companionship and shared experiences have enriched our journey, making it both memorable and meaningful.

ACKNOWLEDGEMENTS

We would like to express our deepest appreciation to our supervisor, Dr. Muaadh Shaif Mohammed Mukred, for his exceptional guidance, support, and wisdom throughout this project. His expertise and encouragement were crucial in overcoming obstacles and enhancing our understanding of Artificial Intelligence and Machine Learning.

We are also profoundly grateful to Dr. Tang Tiong Yew, our subject supervisor, and the distinguished faculty of Sunway University, including Dr. Mikkay Wong Ei Lien and Dr. Fathey Mohammed. Their insights and contributions were instrumental in the successful integration of Business Analytics and AI in this project.

A special thanks to our classmates in the Master of Business Analytics program for their support, friendship, and the shared experiences that have made this academic journey both enjoyable and rewarding.

Finally, we would like to thank Sunway University for providing the necessary resources and a nurturing environment that enabled us to thrive and achieve our academic goals.

ABSTRACT

Automated software testing has become crucial in the digital era, where software reliability and performance are essential. However, traditional manual testing methods often struggle to meet the increasing demand for speed and accuracy. This project addresses the need for an efficient testing system by developing an automated software testing framework for QueueBee's Queue Management System. Therefore, this research aims to enhance the reliability and efficiency of software testing by integrating Selenium WebDriver with Generative AI to automate test execution and reporting. The research methodology involved Software Development Life Cycle (SDLC) to design a prototype using Selenium for test automation and incorporating Generative AI as a proof-of-concept in one test case, alongside performance evaluation against manual testing methods. Performance metrics such as execution speed, accuracy, and scalability were compared between manual and automated testing. A survey was then conducted to determine the company's readiness to implement the automation software testing system. The results showed that the automated testing system significantly improved testing speed while increasing accuracy, productivity, scalability, and consistency as compared to traditional manual testing. This research has contributed towards the development of a robust, scalable automated testing framework and the introduction of Generative AI into the software testing process, offering a more efficient and reliable solution for quality assurance.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Chapter Overview

This chapter sets the stage for this project by introducing the key challenges in software testing, particularly revolving around the Queue Management Systems by QueueBee Solutions. It highlights the limitations of traditional manual testing methods, such as time constraints and human error, and introduces the integration of Selenium WebDriver and Generative AI as a solution. The problem statement underscores the need for a more efficient and comprehensive automated testing system, with Generative AI enhancing the reporting process. The chapter concludes by outlining the research objectives, scope, and significance, providing a clear direction for the study.

1.2 Research Background

1.2.1 Queue Management System

Among various software applications utilized in the current business practices, Queue Management System (QMS) plays a pivotal role in optimizing service delivery and enhancing customer flow management. A QMS is designed to organize and manage queues within retail settings, banks, hospitals, and public service centers. The primary goal of a QMS is to reduce the overall waiting times, streamline customer interactions through the service points while improving efficiency. By managing the flow of customers systematically, businesses can deliver better service experiences, and eventually, leading to higher customer satisfaction.

QueueBee Solutions, a leading provider of queue management systems in Southeast Asia, offers an innovative suite of QMS products tailored for various industries, including finance, healthcare, retail, and public services (QueueBee Solutions, n.d.). QueueBee's solutions go beyond traditional queue management by including digital signage, appointment scheduling, and robust tools to manage different customer touchpoints through a web-based platform. These features enable businesses to optimize their service processes, reduce wait times, and provide a seamless experience for customers.

Despite the significant benefits of using a QMS, ensuring the reliability and performance of the system is crucial. Any failure or inefficiency in the system can lead to prolonged wait times, disrupted service delivery, and the worst case, customer dissatisfaction. For instance, Bidari et al. (2021) observed an inverse relationship between waiting times and patient satisfaction levels in the emergency department of a hospital, in which any delay in the QMS can affect patient satisfaction. Similarly, Mittal (2016) discussed how waiting time affects customer experience in a restaurant setting and suggested that when the wait for a service is perceived by customers as longer than necessary is a source of dissatisfaction and diminished customer loyalty.

The complex nature of a modern QMS application like QueueBee requires thorough and efficient software testing to ensure it is performing as expected under various conditions and user loads. Traditional manual testing methods, while widely used, often fall short due to their time-consuming nature and prone to human error (Dustin et al., 2009). Manual testing may also struggle to adequately cover the unpredictable potential scenarios and inputs that a QMS should handle. The study by Rojas and Arcuri (2015) demonstrated that automatically generated tests tend to have more coverage as compared to the manually written ones. This underscores the need for automated testing solutions, which can provide more comprehensive, accurate, and efficient testing coverage.

1.2.2 Software Testing

In Industrial Revolution 4.0, software programs and applications act as an important catalyst to drive digital transformation across businesses and industries for data exchange and connectivity. The reliance on software extends beyond industrial applications and is essentially in influencing millions across every aspect of modern living, necessitating the need for reliable and error-free softwares. However, the fundamental involvement of human elements in software development is often associated with consequences of human factors such as human error, reduced performance and limited efficiency, emphasizing the need to address software errors or bugs in the development phase (Guveyi et al., 2020). According to a report by the National Institute of Standards and Technology (NIST), software bugs are estimated to cost around $59.5 billion each year in the U.S. economy, but testing efforts can be enhanced to potentially save half of the cost (National Institute of Standards and Technology, 2010). Therefore, it is important to conduct a thorough process of software testing before launching.
Software testing represents an important phase within the Software Development Life Cycle (SDLC) process that encompasses activities that aim to evaluate softwares to identify faults or

errors in its behavior. This evaluation process is conducted to check if the software satisfies all requirements outlined in the design phase, is able to complete the task within the time limit, and produces the correct output according to different inputs and operational environment (Sneha & Malle, 2017). Within the SDLC framework, a software application remains incomplete until all test cases have successfully passed the test. While a test case refers to an action or a set of actions established to verify the functionality of a specific feature of a software application, a test case that does not find an error is often referred to as a “successful” test run (Fraser & Arcuri, 2014). In an ideal world, a simple software program can have hundreds or thousands of test cases due to the large amount of input and output combinations (Myers et al., 2011). Therefore, completing a testing process manually for a complex application is economically infeasible as it can take a very long time and require large amounts of human resources. Notably, the software testing process usually consumes approximately 50% of the time and effort allocated in the software projects, highlighting its substantial role in the software development process (Basak & Hosain, 2014).

1.2.3 Automated Software Testing

Given the challenges and limitations associated with manual testing, automated software testing has emerged as an optimal solution. Automated software testing involves the use of software tools to execute pre-scripted tests on a software application before it is released into production. These tools can run tests repeatedly and consistently, providing a more efficient, reliable, and comprehensive approach to software testing (Dustin, Rashka, and Paul, 1999).

1.3 Problem Statement

While digital transformation is evolving rapidly, businesses across various sectors are increasingly relying on software applications to enhance service delivery and improve operational efficiency (McKinsey & Company, 2024). One of the most critical applications is the Queue Management System (QMS), which plays an important role in organizing and managing customer flow in places such as banks, hospitals, retail stores, and public service centers. QueueBee Solutions offers a comprehensive web-based QMS that includes features for managing queues, digital signage, and customer touchpoints. However, ensuring the reliability and efficiency of a complex system like QMS highlights several challenges, especially in the software testing process (Aniche, 2022).

The traditional manual testing methods that are commonly utilized for software quality assurance often encounter limitations. These methods are time-consuming, prone to human error, and often fail to cover most of the potential user scenarios (Candea, Bucur & Zamfir, 2010). As a result, critical bugs and performance issues may go undetected until after the software is deployed, leading to disruptions in service, customer dissatisfaction, and potential financial losses for businesses using QueueBee's system. Given the essential role of QueueBee in facilitating efficient customer flow management, any system failure can have a substantial negative impact on business operations and customer experiences (Deming et al., 2021).

Moreover, the dynamic and evolving nature of business environments amplify that QMS applications like QueueBee must perform smoothly under varying conditions and user loads (Kiplagat, 2020). This requirement further complicates the testing process, as the system must be rigorously evaluated to ensure it can handle peak loads, varying user interactions, and without compromising performance especially in the era of big data (Tyagi, 2021). Thus, the need for a more efficient, reliable, and comprehensive testing solution is evident. Automated software testing presents a promising approach to address these challenges. By automating the testing process, we can enhance the accuracy, efficiency, and more, significantly reducing the likelihood of undetected issues and improving the overall quality of the software (Ricca, Marchetto & Stocco, 2021).

Therefore, this project aims to develop, evaluate, and assess an automated software testing system for QueueBee while understanding the advantages of implementing an automation system from a business analytics viewpoint. Ultimately, the successful implementation of an automated testing system will ensure that QueueBee can deliver consistent, high-quality service, meeting the needs of its diverse clientele while enhancing customer satisfaction and operational efficiency.

1.4 Research Objectives

- To develop an automated software testing system using Generative AI

The primary objective of this research is to develop an automated software testing system that integrates Selenium WebDriver and Generative AI. Selenium WebDriver will be used to automate a wide range of test cases by simulating user interactions across different browsers and environments (Vila, Novakova & Todorova, 2017). Whereas Generative AI will be used with the implementation of prompt engineering to facilitate test cases (Ooi et al., 2023). This objective aims to develop an automated software testing system to replace QueueBee’s current manual testing system.

- To evaluate the performance between manual testing and automated software testing system

The second objective focused on a comparative analysis of manual testing and the newly developed automated software testing system. The comparison will involve evaluating key performance metrics such as test coverage, execution time, accuracy, and defect detection rates (Bhanushali, 2023). Manual testing that is widely used in the current business practice of businesses is often time consuming and likely to face human error (Sangwatthanarat, 2021).

- To assess the readiness of the company to implement an automated software testing system for business use-case

The final objective of this research is to assess the company's readiness to implement an automated software testing system. This involves conducting a survey with the company’s management to evaluate the current testing infrastructure, available resources and the team’s skill set on handling a newly developed automated testing system. This survey will provide a clear understanding of the company’s readiness and preparations needed for a successful transition to automated testing. (Moodley & James, 2024).

1.5 Research Questions

- How to develop an automated software testing system powered by Generative AI , including technologies such as Generative AI?
- How to select the right metrics to evaluate the performance between automated testing and manual testing?
- How to effectively and accurately assess the company's readiness to implement an automated software testing system and replace manual testing?

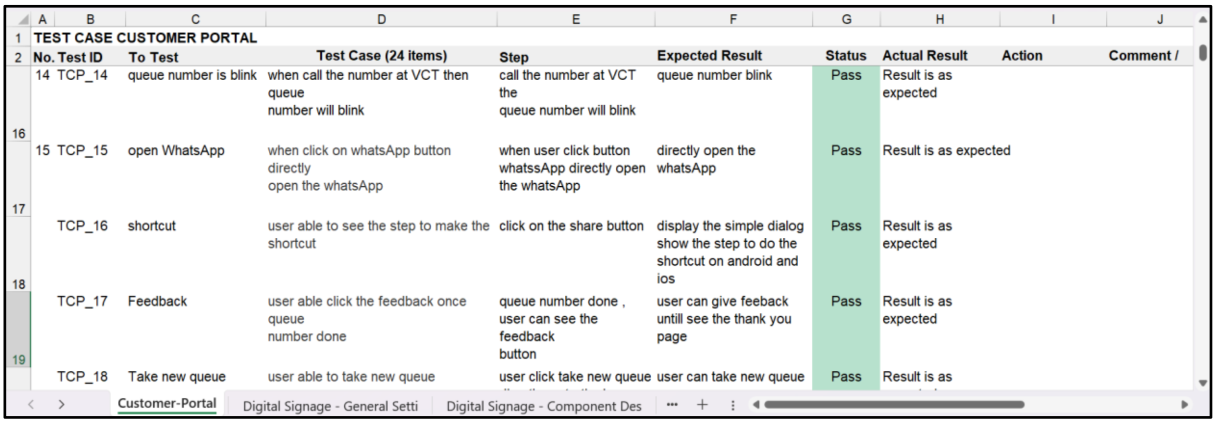
1.6 Research Scope

The scope of this research is designed to ensure a comprehensive analysis and implementation of an automated software testing system. The key components of the scope are outlined to provide an effective framework on achieving the research objectives.

The primary focus of the research involves automating the software testing process with the utilization of Selenium WebDriver as the main tool. Selenium WebDriver is selected based on past research that demonstrates its robustness and capability to simulate user interactions with web applications across various browsers and environments. Studies by Rojas and Arcuri (2015) have highlighted the effectiveness of selenium in providing comprehensive test coverage and identifying defects with high precision, making it an ideal tool for this project. This involves automatically executing the test scripts to ensure thorough testing coverage and high accuracy in identifying potential issues within the software.

On another perspective, the core of the automated software testing will be focusing on the customer portal of the queue management system of QueueBee Solutions. This research is conducted by collaborating with QueueBee Solutions, which the core of the automated software testing will be focusing on the customer portal of the company's product, Queue Management System (QMS). The QMS comprises mainly three components, including the customer portal, admin dashboard, and terminal. However, this research will specifically revolve around automating the testing of the web-based customer portal to ensure its reliability and performance.

To facilitate the automated testing process, QueueBee has provided detailed test scripts, as shown in Figure 1, that cover various functionalities and user interactions within the customer portal, specifically revolving the User Interface (UI). The execution of these test scripts will be carried out automatically using selenium to ensure a comprehensive testing coverage. Furthermore, it is crucial that these test scripts are effectively used and maintained for continuous testing and reliable performance of the customer portal.



TEST CASE CUSTOMER PORTAL

No.	Test ID	To Test	Test Case (24 items)	Step	Expected Result	Status	Actual Result	Action	Comment /
14	TCP_14	queue number is blink	when call the number at VCT then queue number will blink	call the number at VCT the queue number will blink	queue number blink	Pass	Result is as expected		
15	TCP_15	open WhatsApp	when click on whatsApp button directly open the whatsApp	when user click button whatssApp directly open the whatsApp	directly open the whatsApp	Pass	Result is as expected		
	TCP_16	shortcut	user able to see the step to make the shortcut	click on the share button	display the simple dialog show the step to do the shortcut on android and ios	Pass	Result is as expected		
	TCP_17	Feedback	user able click the feedback once queue number done	queue number done , user can see the feedback button	user can give feeback untill see the thank you page	Pass	Result is as expected		
	TCP_18	Take new queue	user able to take new queue	user click take new queue	user can take new queue	Pass	Result is as		

Customer-Portal
Digital Signage - General Setti
Digital Signage - Component Des

Figure 1: A snapshot of the test cases of the customer portal

By addressing these key components, this research aims to develop a robust automated software testing system that enhances the efficiency and accuracy of the testing process. The successful implementation of this system will ensure that QueueBee's customer portal operates reliably under various conditions, ultimately improving customer satisfaction and operational efficiency.

1.7 Significance of the study

This research holds significant importance in both practical and theoretical aspects of software testing within the context of modern business operations.

First and foremost, this study represents a critical advancement in the field of software testing technology. By automating the software testing process, this research addresses the limitations of traditional manual testing methods, such as inefficiency, human error, and limited coverage. Thereby, a more reliable and comprehensive testing outcome can be ensured. The technological contribution of this study lies in its ability to enhance the precision and scalability of software testing by utilizing Selenium and Generative AI, which is crucial in an era where software systems are becoming increasingly complex and integral to various industries. Hence, this study sets a new standard for the industry by demonstrating the edges of automated software testing over traditional methods to encourage wider adoption of automated solutions that can significantly improve the quality and reliability of software applications.

From a business analytics perspective, the implementation of such automated software testing systems is critical. In this era of digitalization, the reliability of business systems are undeniably important as companies are increasingly relying on software applications to drive efficiency and customer satisfaction. By reducing the likelihood of software failures through enhanced testing accuracy and efficiency, this research contributes to higher operational standards across industries, leading to improved service delivery and customer trust. Moreover, companies that adopted such automated systems can achieve faster time-to-market, reduce operational costs, and maintain a competitive edge, further emphasizing the contribution of this research. Therefore, this project does not only impacts the technology behind software testing but also plays a crucial role in supporting business innovation and resilience in an increasingly digital world.

CHAPTER 2

LITERATURE REVIEW

2.1 Chapter overview

This literature review provides an in-depth exploration on automated software testing, generative artificial intelligence, and the application of business analytics in software testing. The chapter begins with a glossary that defines essential terms and concepts to ensure clarity for the reader. Following this, the chapter delves into the facets of automated software testing, addressing its key components: efficiency, accuracy, coverage, repeatability, integration with CI/CD pipeline, and quality measurement. The evolution of Generative AI and its application in software testing is explored next. The approaches utilized to measure key factors related to software testing and its impact on business decisions is examined by discussing business analytics in software testing. Lastly, three major studies that employ automated software testing approaches are discussed in detail. Their methodology, key findings, and arguments are scrutinized to gain insights regarding automated software testing. The findings from the 3 studies are then synthesized in a critical analysis to offer a cohesive understanding of the automated software testing landscape.

2.2 Glossary

This section includes all the glossaries mentioned in this paper.

Term	Definition
Artificial Intelligence (AI)	Ability of machines to simulate human intelligence.
API	Application Programming Interface
Automation	Use of machines to perform tasks without human intervention.
Bugs	Unexpected problem with the software or hardware.
CI/CD	Continuous Integration / Continuous Development.
Chatbot	Software that simulates human conversation with a user.
GraphWalker	Model-based testing automation tool.
Katalon Studio	Software tool for automated testing.
Prompt Engineering	Process of structuring an instruction that can be interpreted by an AI model.
Revenue	Money generated from business operations.
Selenium	Tools and libraries to support browser automation.
SMS API	A tool to integrate messaging into your existing software.
Test Cases	Specification of test procedure, inputs, and expected results.
Test Project	Automation platform for web, mobile, and API.
Test Script	Step-by-step instructions on how to test a software.

2.3 Automated Software Testing

Automated software testing is a development in the software testing field that has changed the way software testing is conducted. It can be defined as “The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated testing tool” (Dustin et al., 1999). Manual software testing solely depends on human efforts, becoming time consuming and expensive in the long run (Halani et al., 2021). With the intention to do more in less time, manual testing was paired with automation (Sabev & Grigorova, 2015). The test scripts would still be governed by developers however automation would assist in development and execution of them. Automation has brought in several benefits and improved software testing by improving product quality, reducing testing time, improving fault detection, and more (Rafi et al., 2012). However, automated testing cannot be considered as a perfect solution to software testing and comes with its own set of limitations. These are difficulties in maintaining the system, require skilled people, and wrong expectations held by testers or firms. (Rafi et al., 2012). Automated testing also does not have the capabilities to completely replace manual testing as they cannot be suitable for all areas (Dustin et al., 2009).

According to the book "Automated Software Testing: Introduction, Management, and Performance" by Dustin, Rashka and Paul (1999), the key benefits to implementing automated software testing can be categorized into 6 perspectives, which are efficiency, accuracy, coverage, repeatability, integration with Continuous Integration/Continuous Deployment (CI/CD), and quality measurement and test optimization.

2.3.1 Efficiency

Automated testing can be conducted rapidly with minimum human intervention while allowing the ability to quickly identify defects or errors in the software (Rafi et al., 2012). This improvement speeds up the whole testing process, and therefore, enables more frequent testing cycles and faster feedback (Dustin et al., 1999). By automating repetitive tasks, resources can be better utilized, which software testers can be allocated to focus on more complex aspects of the project (Dustin et al., 2009). The authors also emphasize that this efficiency not only reduces the time to market but also cuts down costs associated with manual testing efforts.

2.3.2 Accuracy

From another perspective, the writers also underscore accuracy provided by automated tests. Manual testing is usually prone to human error, such as missing steps or incorrect data entry (Halani et al., 2021). However, automated testing is able to execute predefined scripts precisely, ensuring consistent and accurate results every time (Albarka & Zhanfang, 2019). This consistency enhances the reliability of test outcomes and reduces the likelihood of defects slipping through the cracks.

2.3.3 Coverage

The authors also discussed how automated testing extends test coverage beyond what is feasible with manual testing. Automated tests can cover a wide array of scenarios, including complex conditions (Dustin et al., 1999). This extensive coverage ensures that more parts of the application are tested, leading to the discovery of errors which manual testing may not disclose (Rafi et al., 2012). The book further provides insights into designing comprehensive automated test suites that address both functional and non-functional requirements.

2.3.4 Repeatability

Repeatability is another critical benefit highlighted in the book. Automated tests can provide consistent results even being re-run multiple times with the same input and expected output (Dustin et al., 1999). This repeatability is essential for regression testing, where previously validated functionalities need to be re-tested after any code change to ensure no new defects are introduced into the software (Sutapa et al., 2020). The authors discuss the importance of maintaining a robust set of regression tests to ensure software stability and reliability.

2.3.5 Integration with CI/CD Pipelines

Moreover, automated testing can be integrated seamlessly into the continuous integration and continuous deployment (CI/CD) pipeline, enabling continuous testing and immediate feedback as code changes (Dustin et al., 1999). This integration ensures that any issues are identified and addressed promptly during early stages of development to prevent risk of introducing faulty code into production (Sane, 2021). Therefore, the agility and reliability of SDLC can be enhanced.

2.3.6 Quality Measurements and Test Optimization

The book also addresses the importance of quality measurements and test optimization through automation. Automated testing tools can collect detailed metrics on test execution, such as pass/fail rates, execution time, and defect density (Dustin et al., 1999). These metrics play an important role to provide valuable insights into the software's quality and the effectiveness of the testing process (Sneha & Malle, 2017).

2.4 Generative Artificial Intelligence

Generative Artificial Intelligence (AI) was first introduced in the 1960s, being implemented in chatbots (Kılınç & Keçecioğlu, 2024). However, this can be considered primitive when compared to the technology present today. The term 'Generative AI' refers to a set of computational techniques that are capable of generating new content from training data, such as text, images, videos, and more. (Feuerriegel et al., 2023). It has revolutionized various fields, such as computer vision, natural language processing, and creative arts (Bandi et al., 2023). Few decades ago, the advancement of AI was stagnant due to the lack of processing capacity and spending (Nah et al., 2023). However in the last few years, AI has made a comeback, especially with the launch of ChatGPT that is powered by Generative AI (Nah et al., 2023). Refer to Figure 2 below which illustrates the evolution of Generative AI from 1950 to 2022.

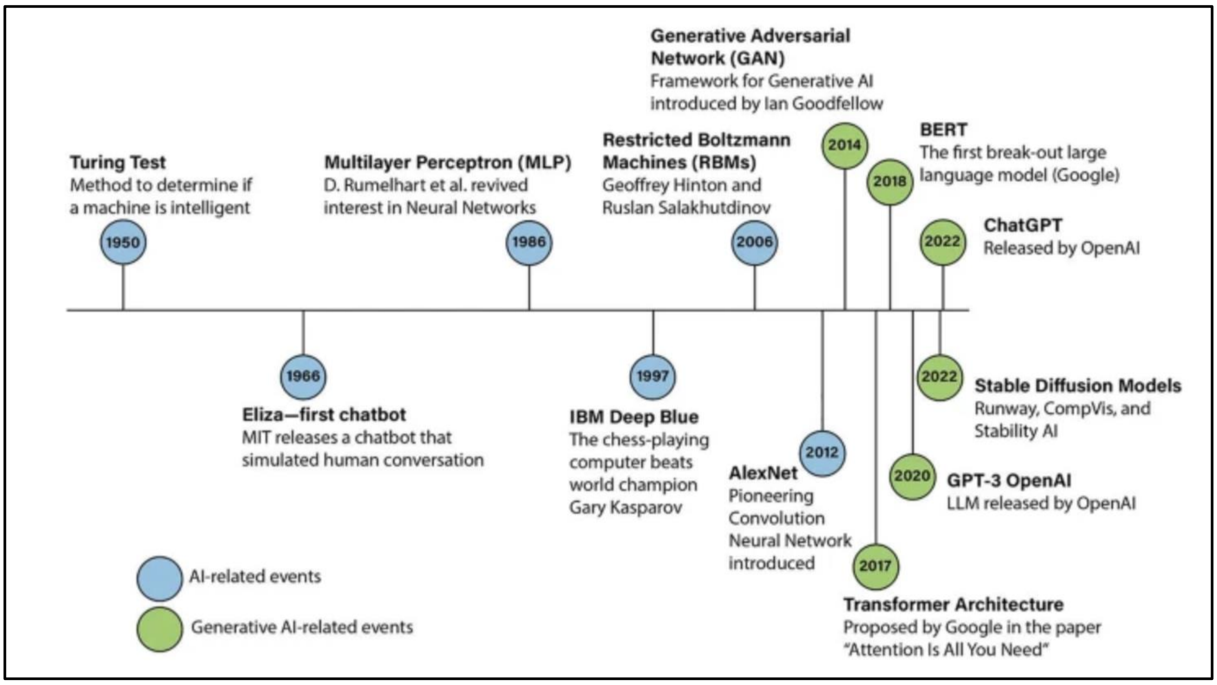
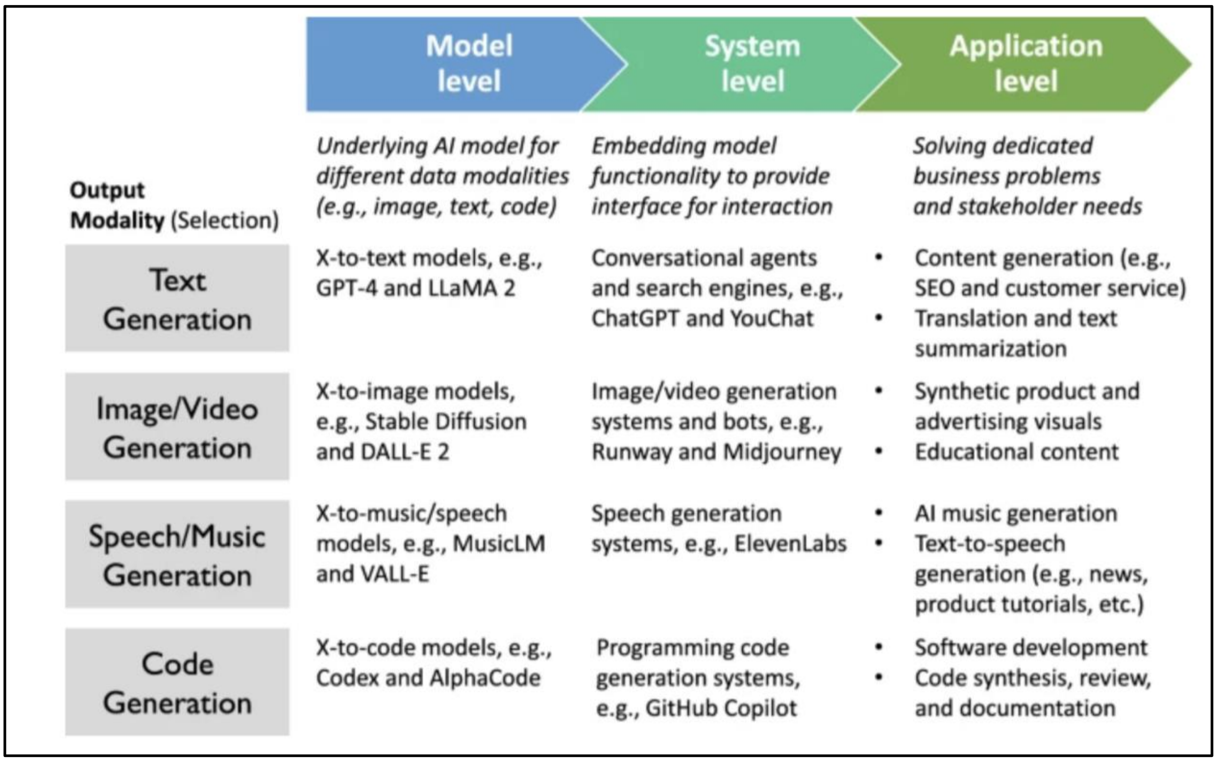

Generative Adversarial Network (GAN)
Framework for Generative AI introduced by Ian Goodfellow
BERT
The first break-out large language model (Google)
Restricted Boltzmann Machines (RBMs)
Geoffrey Hinton and Ruslan Salakhutdinov
Turing Test
Method to determine if a machine is intelligent
Multilayer Perceptron (MLP)
D. Rumelhart et al. revived interest in Neural Networks
ChatGPT
Released by OpenAI
2014
2018
2022
1950
1986
2006
1966
1997
2012
2022
2020
2017
Eliza—first chatbot
MIT releases a chatbot that simulated human conversation
IBM Deep Blue
The chess-playing computer beats world champion Gary Kasparov
AlexNet
Pioneering Convolution Neural Network introduced
Stable Diffusion Models
Runway, CompVis, and Stability AI
GPT-3 OpenAI
LLM released by OpenAI
Transformer Architecture
Proposed by Google in the paper "Attention Is All You Need"
AI-related events
Generative AI-related events

Figure 2: Evolution of Generative AI (Pulapaka et al., 2024)

Generative AI is built and based on generative modeling that gathers information from a data distribution (Feuerriegel et al., 2023). Hence, a generative AI model is a generative model represented by a machine learning algorithm and can create new data based on learned samples (Kenthapadi et al., 2023). The generative AI system consists of all the components, the model, the data processing, and the user interface components (Tomczak, 2024). A generative AI application is the practical use case and implementation of this system, such as blog content generation or code generation (Weisz et al., 2024). Refer to Figure 3 below to view a summary of the model level, system level, and application level view on generative AI.



Output Modality (Selection)	Model level	System level	Application level
	Underlying AI model for different data modalities (e.g., image, text, code)	*Embedding model functionality to provide interface for interaction*	*Solving dedicated business problems and stakeholder needs*
Text Generation	X-to-text models, e.g., GPT-4 and LLaMA 2	Conversational agents and search engines, e.g., ChatGPT and YouChat	• Content generation (e.g., SEO and customer service) • Translation and text summarization
Image/Video Generation	X-to-image models, e.g., Stable Diffusion and DALL-E 2	Image/video generation systems and bots, e.g., Runway and Midjourney	• Synthetic product and advertising visuals • Educational content
Speech/Music Generation	X-to-music/speech models, e.g., MusicLM and VALL-E	Speech generation systems, e.g., ElevenLabs	• AI music generation • Text-to-speech generation (e.g., news, product tutorials, etc.)
Code Generation	X-to-code models, e.g., Codex and AlphaCode	Programming code generation systems, e.g., GitHub Copilot	• Software development • Code synthesis, review, and documentation

Figure 3: Model, System, and Application Level View of Generative AI (Feuerriegel et al., 2023)

Generative AI has been utilized in software testing as well with its applications ranging from automating testing to test case generation. A previous study created a chatbot powered with Generative AI to successfully develop test cases and scenarios and generate them automatically (Garg & Sharma, 2023). Similarly, in another study, Generative Adversarial Networks (GAN) were used to improve testing efficiency by automatically generating high quality test cases (Guo et al., 2022). Another study utilized ChatGPT in the entire software development process, which included generating test cases automatically to test the software (Rajbhoj et al., 2024). The use of Generative AI in software development is still being researched further and not yet explored fully.

2.5 Business Analytics in Software Testing

Business Analytics (BA) refers to the application of analytical concepts to business activities, such as creating marketing strategies, preparing revenue forecasts, or identifying customer segments (Power et al., 2018). This concept is a compound noun as it is two independent concepts merged together, ‘business’ and ‘analytics’ (Power et al., 2018). The term ‘analytics’ can be defined as the processes used to transform raw data into meaningful insights to better decision making (Wilder & Ozgur, 2015). While this concept is not new, it has recently made a comeback due to the capabilities it possesses to handle Big Data (Schwarz et al., 2014). The revolution of BA has impacted several businesses and fields however a huge amount of data, computer processing power, and large storage devices are required to effectively implement BA (Acito & Khatri, 2014). Refer to Figure 4 below which illustrates a structural framework for acquiring value from BA. It requires aligning the firm’s strategy and desirable behaviors to business performance management and simultaneously with analytical tasks and capabilities (Acito & Khatri, 2014).

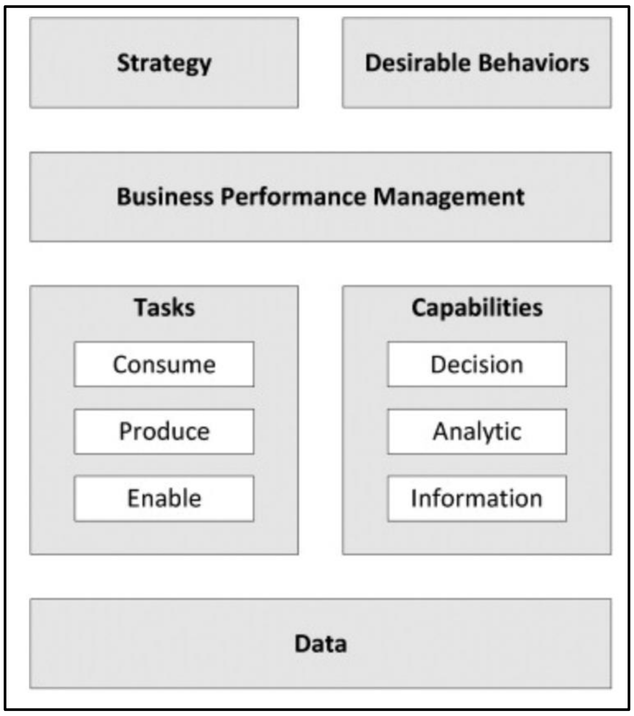


Figure 4: Business Analytics Structural Framework (Acito & Khatri, 2014)

In the case of software development and testing, BA provides several applications to improve the efficiency and effectiveness. While some professionals make strategies based on ‘gut feeling’ or past experiences, research has shown that data-driven decisions lead to better business performance (Tuboalabo et al., 2024). This led to the birth of software analytics, a branch of big data analytics, which focuses on visualizations and data interpretation to aid decision-making in software development (Abdellatif et al., 2015). Software testing is a procedure done to ensure the software program or application runs without any bugs or errors (Sneha & Malle, 2017). To improve efficiency and productivity, automated testing tools have been aiding software testers and delivering projects in a shorter time (Kumar & Rao, 2012). Software analytics and BA are able to supplement software testing by providing useful interpretations of the software testing data. It begins with problem identification and the insights gained are used to improve the software development or software testing phase (Bener et al., 2015). Efficiency and accuracy is extremely important in the software testing phase as it is one of the most time-consuming and important stages of software development (Olaleye et al., 2023). Predictive analytics is also used in creating software defect predictive models that can be used to predict any defects by using data science (Kamal et al., 2022). This can be used to avoid costly defects in the future and ensure users do not face critical issues (Olaleye et al., 2023). Moreover, machine learning algorithms are also being widely used in software testing to automate testing and bug detection (Mahapatra & Mishra, 2020; (Tanaka et al., 2019). This improves a firm’s software development cycle and reduces the tester’s workload, saving time and money. Business analytics can also be leveraged to improve a firm’s innovation as a recent study has found that BA enhances a data-driven culture and environmental scanning, which improves strategy creation and innovation (Duan et al., 2020).

2.6 Existing Solutions of Automated Software Testing

2.6.1 Test Generation using Large Language Models

This study is titled 'Effective test generation using pre-trained Large Language Models and mutation testing' and is conducted by Dakhel et al. (2024).

- Methodology

Large Language Models (LLMs) have been utilized to generate test cases for software testing. This study has created a prompt-based learning technique called MuTAP (Mutation Test case generation using Augmented Prompt) to improve the efficacy of the test cases generated by LLMs. Different LLMs were deployed under MuTAP to assess and evaluate their performance on different benchmarks. The goal of MuTAP is to discover and highlight faulty bugs in the codes of the test cases to improve automated software testing. Refer to Figure 5 to view the suggested methodology for the generation and evaluation of test cases using LLMs.

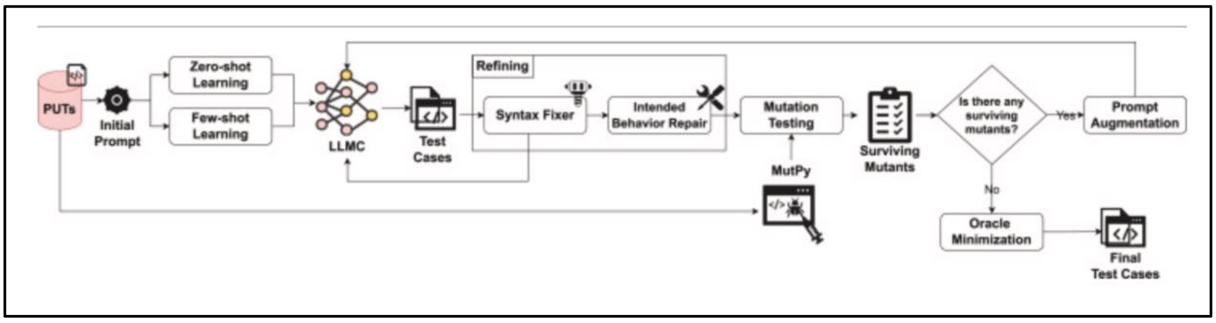


Figure 5: The suggested methodology for generating and evaluating test cases using LLMs

- Key Findings

Main findings	**Explanations and justifications**
Automatic Generation of Test Cases	The findings showed that LLMs produce more natural test-cases with better functionality when compared to Pynguin (an advanced automated test-case generation tool). However, they require post-processing to be more successful in finding bugs.
Prompt Engineering for Automated Test Case Generation	The study found several key insights related to prompt engineering. Firstly, using a fixed function name in the zero-shot prompt and allowing LLMs to determine this function name did not affect the effectiveness of test cases in revealing bugs. Hence, both methods of choosing an identifier name for the test case are effective. Secondly, the prompt is also able to instruct the LLM component to generate a test case with the capability of killing the surviving mutant. Hence, it can find the bug's specific line and create a new test case to detect that difference. Including the information of the mutant in the prompt can further improve the effectiveness of the prompt augmentation.
Execution Time	The processing time was shorter using zero-shot prompting with an average of 39.75 seconds compared to few-shot learning, which had an average of 42.11 seconds. However, the difference is small and both prompts are effective.
Benefit of a dialog LLM	Including a dialog llama 2 chat allowed flexibility in assigning roles of each component of the augmented prompt which reduced the likelihood of repeating the initial test cases in the generated output.

Table 1: Key Findings for Study 1

- Arguments

Several key arguments were discovered from this study. Firstly, the tool created by the researchers called MuTAP was proposed as an effective tool in enhancing LLMs for effective test case generation. It is also able to evaluate the test cases and its effectiveness by measuring how well it detects bugs. Secondly, MuTAP was found to have better bug detection rates when compared with traditional automatic test generation techniques and other LLM techniques. Lastly, it is adaptable to several different LLMs, showing versatility. Future work could focus on its adaptability with different programming languages, new LLMs, and investigate which prompt structure would be best to enhance test cases.

2.6.2 Model Based Testing

This study is titled 'Web Application Testing With Model Based Testing Method: Case Study' and is conducted by Akpinar et al. (2020).

- Methodology

This study investigated the differences between model-based software tests and traditional tests. These two methods were compared and reported on parameters of code line number, actual error detection rate, and test run time. The tests for the model test method were developed using the Selenium library with the GraphWalker tool. The model-based software testing was applied to the Web user interface of Loadium, which is a load test automation system. On the other hand, the traditional tests were manually prepared using JUnit and TestNG libraries. Both testing methods were compared by running them several times to collect data on text execution, error detection, and code complexity. Refer to Figure 6 below for the framework of the model-based testing.

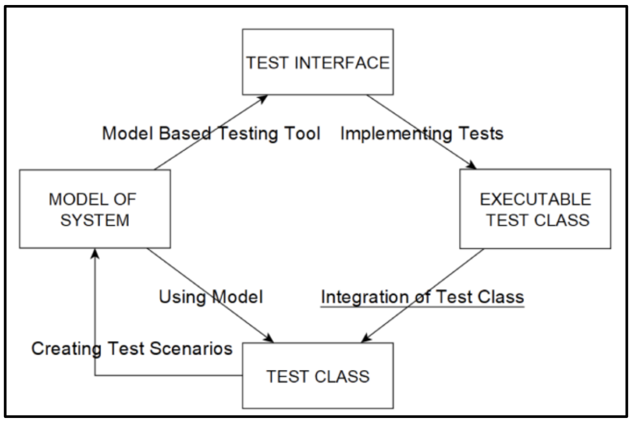
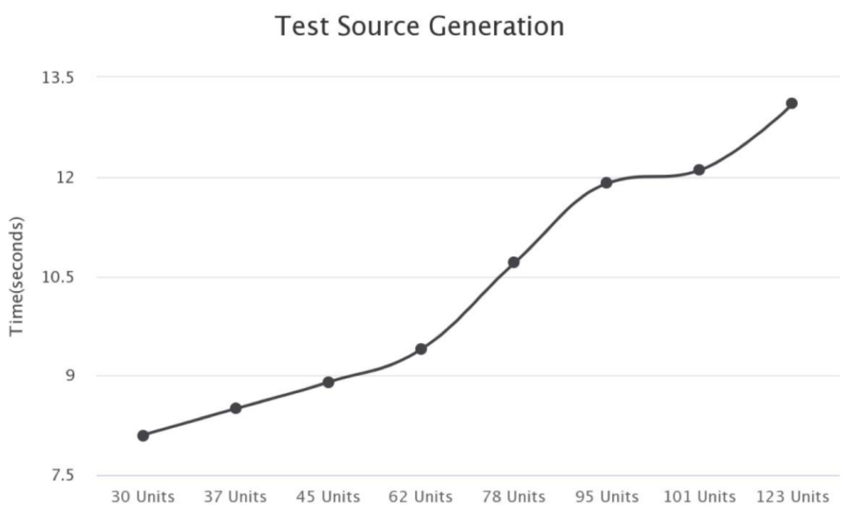
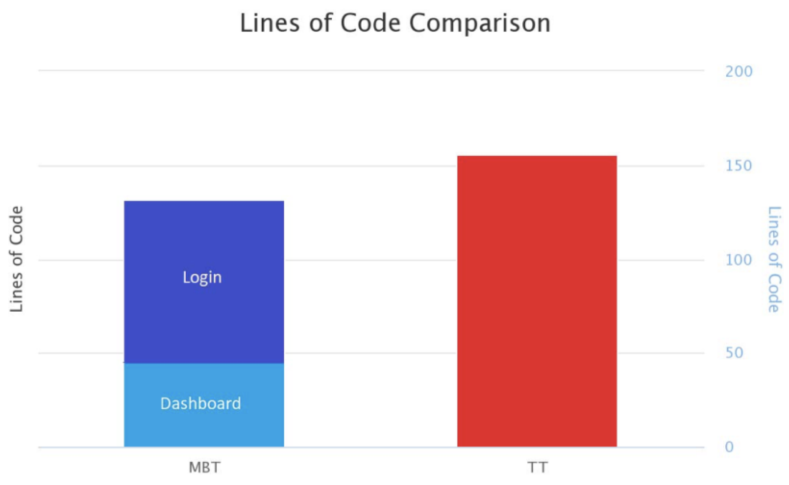


Figure 6: Framework of the model-based testing

- Key Findings

Main findings	**Explanations and justifications**
Coverage Rate and Test Process	The study found that model-based tests showed higher coverage rates and faster test processes compared to traditional tests. GraphWalker, which is used in the model-based method, is able to generate random paths when the test runs, ensuring all nodes and edges are crossed. This allows it to have a more extensive coverage than a traditional test, where it is difficult for an average test developer to do this manually.
Test Developer Workload	The model-based test method was able to significantly reduce the test developer's workload by automating test scenario generation. This was evaluated by measuring the test source generation and lines of code. The study found that the model could generate test cases of high complexity and require fewer lines of code than the traditional test method, reducing workload. Refer to Figure 7 below to view GraphWalker's test generation time which shows high coverage of units without a steep increase in time. Refer to Figure 8 below to examine the workload comparison between model based testing and traditional testing, measured by lines of code

	 *Figure 7: Test Source Generation according to summation of vertices and edges count (Unit)*  *Figure 8: Comparison graph of lines of implemented code*
Error Detection	The study discovered that model-based tests had a similar detection rate to traditional tests. However, they are more effective in finding errors about page transitions and repeated navigation, something which traditional tests are less likely to cover.
Execution Time	The researchers found that the average execution time for model-based tests was shorter than traditional tests. This was due to the automated generation and execution of test scenarios possible in

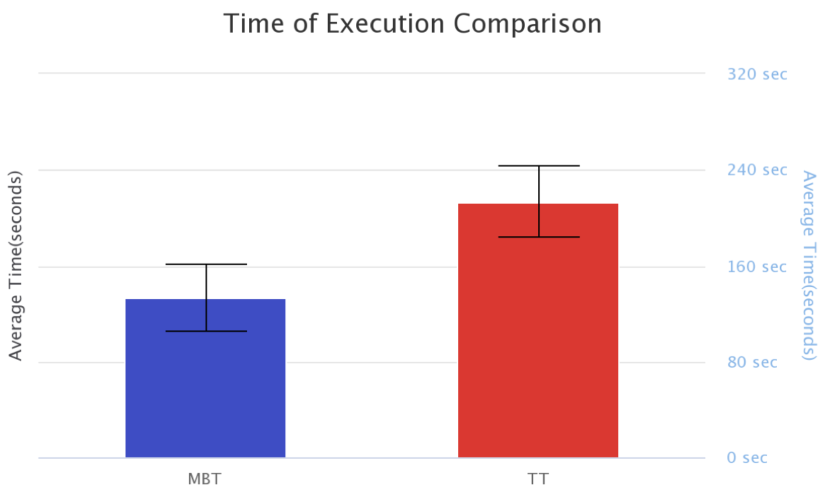
	model-based testing. Refer to Figure 9 below to view the comparison of test execution time between model based testing and traditional testing.  *Figure 9: Comparison graph of test execution time*

Table 2: Key Findings for Study 2

- Arguments

Several key arguments were discovered from this study. First, the test performance was better in model-based testing as it reduced test developer workload and shortened test times. Second, model-based testing was effective in generating realistic test runs by simulating multiple user interactions. Last, a more comprehensive testing approach was provided by model-based testing as it covered a wider range of scenarios through automated test generation. Future work could evaluate the testing methods in more complex systems, examine metrics using different approaches, and test different testing tools to compare their performance.

2.6.3 Smart Queue Management System

This study is titled 'IoT-Based Smart Queue Management System for Effective Social Distancing and Contact Tracing' and is conducted by Majeed et al. (2021).

- Methodology

This study investigated the performance and features of three open-source automation testing tools: Selenium, Katalon Studio, and Test Project. The research aimed to compare these tools based on their capabilities, ease of use, and overall effectiveness in automating software testing. Key parameters for comparison included setup and configuration time, test execution speed, error detection rate, and community support. The study involved practical experiments where identical test scenarios were executed using each tool to gather comparative data.

- Key Findings:

Main findings	**Explanations and justifications**
Capability of Script Generating and Script Reusability	Selenium offers great flexibility as it supports a wide range of scripting languages (Java, C#, Ruby, PHP, Python, Perl). Katalon Studio supports Groovy and Java, while Test Project uniquely supports Java and C# within a single test case. All three tools support script reusability. Selenium's reusability is highly flexible but requires programming skills. Katalon Studio and Test Project have a more user-friendly interface, offering easier script modification and reuse.
Test Execution Speed	In terms of text execution speed, Selenium showed the fastest speed due to its lightweight nature and direct interplay with the web drivers. Katalon Studio and Test Project were a bit slower but offered more comprehensive features and built-in functions that made test creation and execution smoother.
Error Detection Rate	All three tools were found to be effective in error detection however they used different approaches. Selenium’s approach was highly customisable and it depends on the tester’s expertise. Katalon Studio and Test Project used more user-friendly interfaces for identifying and resolving issues, which is more suited for testers with less programming experience.
Community Support and Team Collaboration	A significant factor found was community support which highly impacted the usability of these tools. As Selenium was the oldest and most widely used, it had the largest community with extensive documentation on the tool. The other two tools had smaller but rapidly emerging communities, with active support forums and some collaborative features. In terms of team collaboration, Selenium and Katalon Studio are limited to single-machine usage making them not suitable for collaboration. Test Project

	is more suitable for distributed teams as it supports team collaboration with remote access capabilities.
Cost	Selenium is completely free of charge and an open-source software. Katalon Studio offers a free version with advanced features available for a certain fee. Test Project is free of charge as well with full functionality for web, mobile, and API testing.

Table 3: Key Findings of Study 3

- Arguments

This study found that all three tools are suitable, depending on the user's expertise and needs. Firstly, Katalon Studio and Test Project are more suited for testers with limited programming knowledge as they are more user-friendly and accessible. Selenium is more suited for its flexibility and customization as it supports multiple programming languages. Hence, it is more ideal for experienced testers that need detailed control over their testing processes. Secondly, in terms of cost, Selenium is completely free and open-source, making it cost-effective for skilled teams. Katalon Studio offers a balance between free and paid advanced features, and Test Project remains free with robust capabilities. Lastly, Test Project stands out for its team collaboration support, beneficial for distributed teams, unlike Selenium and Katalon Studio which are more restricted in this aspect. Hence, the choice of tool should depend on the specific needs and resources of the testing team, with Selenium being suitable for those requiring flexibility and speed, and Katalon Studio and Test Project catering to those seeking ease of use and comprehensive features with minimal setup effort.

2.7 Critical Analysis

As shown from the studies above and other research, Selenium Webdriver is found to be an effective tool for automated testing. The first study demonstrated a strong bug detection tool, called MuTAP, developed by the researchers. It worked better than other automatic and LLM models however it is mainly applicable for bug detection. Hence, it showcases that customizing the model according to the project's needs can provide superior results than the general models. The second study evaluated model-based testing with traditional testing, finding model-based to be superior by reducing workload, generating realistic test runs, and having a wide test coverage. The model-based tests were developed using the Selenium library, showing it to be an effective tool for test-case generation. The third study evaluated different automated testing tools, concluding that all tools are suitable but Selenium offers higher flexibility and is free of cost. Selenium also has a supporting and welcoming community present online, making it easy to solve challenges (Leotta et al., 2023). All of these studies ultimately evaluate automated testing and tools showcasing that it is an effective way but there is no one size fits all as the right tool or model must be chosen. Using automated testing in conjunction with manual testing is also the optimum way, especially to ensure the quality and accuracy of the testing (Horalek et al., 2023). Whilst there is a multitude of research on automated testing and Selenium frameworks, research is still lacking on Generative AI and SMS API. This limitation is present in most research and with the advancement in these technologies, exploring these two areas would be beneficial to improve software testing.

CHAPTER 3

METHODOLOGY

3.1 Chapter Overview

Based on the methodology, it explains the research design to implement the methodology. By giving an understanding of the research design, it forms the research approach of the methodology. With the current problem, the goal would be to develop software automation for the client therefore it takes a pragmatic approach towards the research design as the client focuses more on the implementation of the software. Hence, it requires contextualized knowledge towards building it and there are three methodologies to achieve the research objective.

3.2 Research Design

Towards formulating the base of the research design, Creswell & Creswell (2018) summarize the intertwined relationship between Philosophical worldviews, design and research methods in relation to the methodology of the research approach, which is displayed in Figure 10.

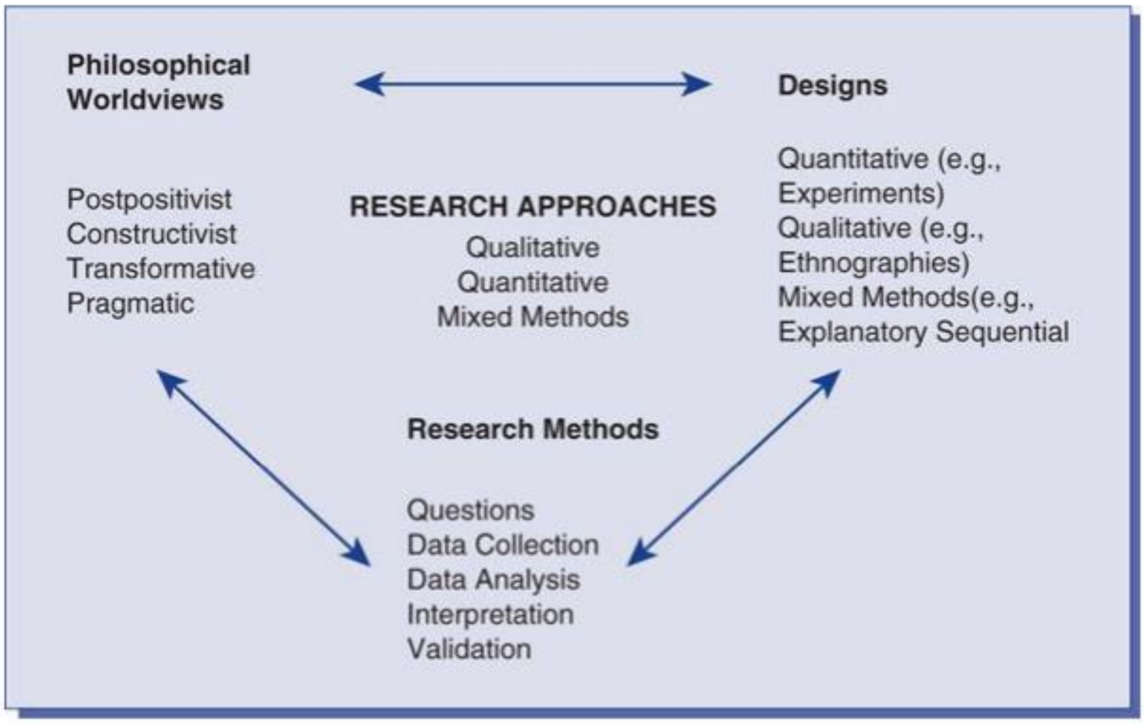


Figure 10: Research Paradigm (Creswell, & Creswell, 2018)

3.2.1 Philosophical Worldviews (Research Paradigm)

Based on Creswell & Creswell (2018), the semantic meaning of worldviews refers to the belief held by individual researchers, therefore guiding the research methodology (qualitative, quantitative or mix-method). Therefore, this section will introduce the four commonly used research paradigms.

- Postpositivist

Derive from the traditional research method known as the positivism (scientific method). Positivism assumes the current knowledge as an absolute truth, where it utilizes a Hypothetico-Deductive model to refine and access theory within the literature reviews (Park, Konge & Artino, 2020). However, implying an absolute truth of knowledge does present limitations. Discrepancy emerges with the emergence of new evidence, challenging the notion of absolute truth. Therefore, postpositivism was introduced to provide the flexibility needed and allowed nuance interpretation towards knowledge (Creswell & Creswell, 2018).

- Constructivist

Yong et al (2023) argue that constructivism was developed to address the limitation of positivism by stating that *'human beings cannot be explored similarly as a physical phenomenon'*. Thereby, introducing a paradigm which focused on qualitative approach (interview and survey) towards research. To further emphasize, social constructivists were built on the foundation that each individual developed opinion subjectively based on their experiences. Therefore, although several individuals may experience similar events, their opinion differs. Hence, the researcher would utilize the approach to develop a theory by identifying a pattern from the participant (Creswell & Creswell, 2018).

- Transformative

Transformative paradigm follows a constructivist narrative which emphasizes the subjective experiences (opinion) of marginalized individuals such as disability, racial etc which are excluded from academia. Building on the constructivist concept of collecting data directly from individuals, the transformative approach underscores the traditional qualitative data collection methods towards certain demographic groups therefore introducing biases. Therefore it overlooks the nuanced experiences of marginalized groups (Creswell & Creswell, 2018).

- Pragmatic

The evolution of pragmatic forms derives from the limitations of positivism, which emphasizes consequences rather than antecedents. The focus of Positivism is built around a theoretical framework narrative, replicating generalizations, valuable for broad understanding and further theoretical experimenting. However, in the context of business problems, it often lacks the contextualized knowledge required for practical application. The implication of pragmatic takes forms as experiential empiricism are incorporated to build more relevant and actionable research towards the business problem. For example, while positivism might be used to identify general trends, a pragmatic approach would delve into specific case studies to understand the unique factors influencing the trends in different markets to build localisation strategies (Ruwhiu & Cone, 2010).

3.2.2 Research Paradigm Methodology Selection

The researcher is currently collaborating with QueueBee to develop an automated software system. The primary goal is to improve productivity by incorporating a software testing system to execute the growing number of test cases. With the number of test cases increasing on every update, it has become unfeasible to manually execute the growing number of test cases. The research would adopt a pragmatic research paradigm methodology as the focus revolved on the practical outcomes of the software (Melegati & Wang, 2021).

3.3 Software Development Life Cycle (SDLC)

The development and maintenance of software systems refers to a series of procedures known as Software Development Life Cycle (SDLC). Although the waterfall methodology was the initial model developed which proves to be robust, it does reflect several limitations such as hardware dependencies and sequential progression. With the development of SDLC throughout the decades, two branches were derived from the initial methodology and categorized as heavyweight and lightweight methodologies. Figure 11 below shows the methodologies mentioned by Ben-Zahia & Jaluta (2014).

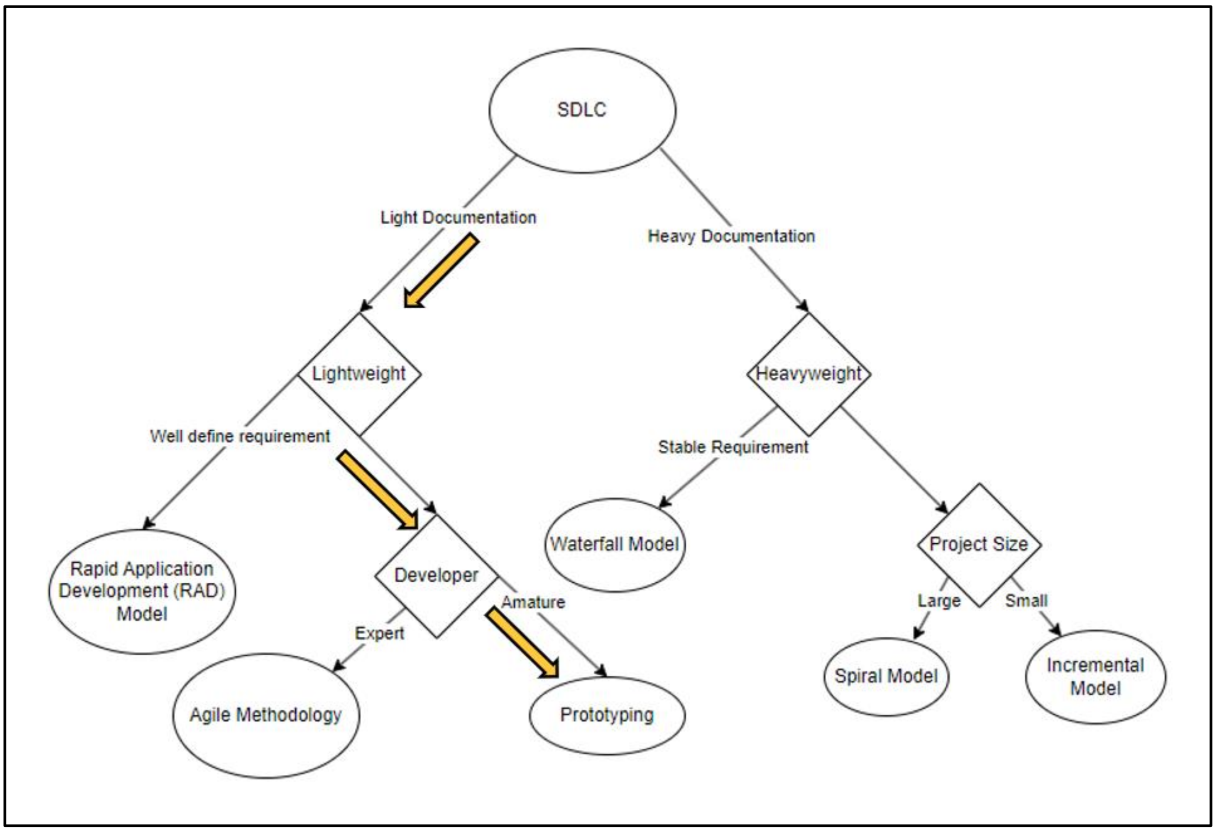


Figure 11: Selection of SDLC

3.3.1 Selected Prototyping Model

Based on the overview of the SDLC image seen above, the project will implement a Prototyping model towards the development of the automation software. The methodological steps are included in the following section (Zhang, Song & Kong, 2004) in Figure 12.

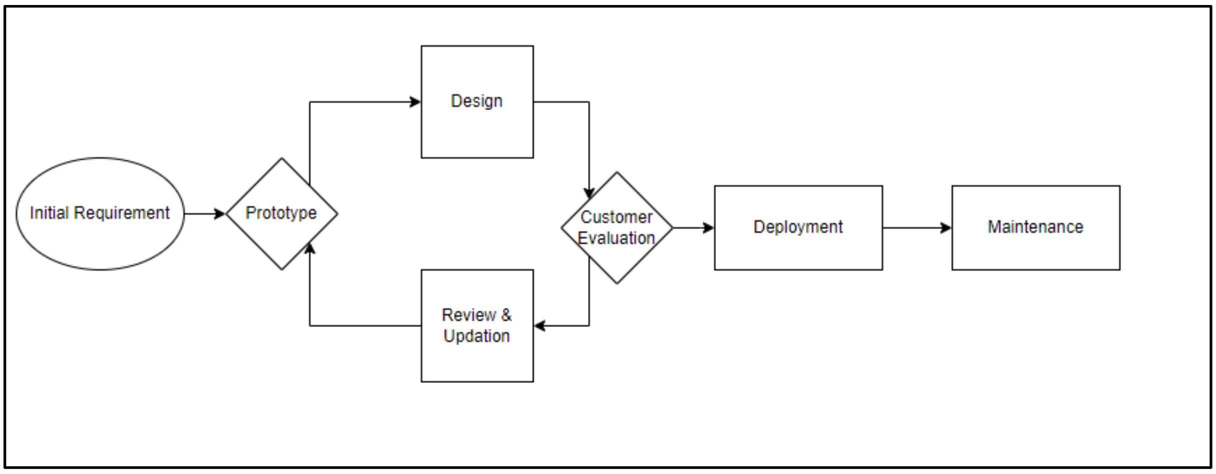


Figure 12: Flowchart for Prototyping model (Pomberger et al, 1998)

3.3.2 Step 1: Initial Requirement

Based on the initial requirement of the software, the current scope of the SDLC would be to develop an automation software system as a prototype and utilize analytical tools to provide insight regarding the software system to the stakeholders, which are the QueueBee (the company involved in this research) and Sunway University (the author's university conducting this research). Refer to Table 4 below for detailed requirements.

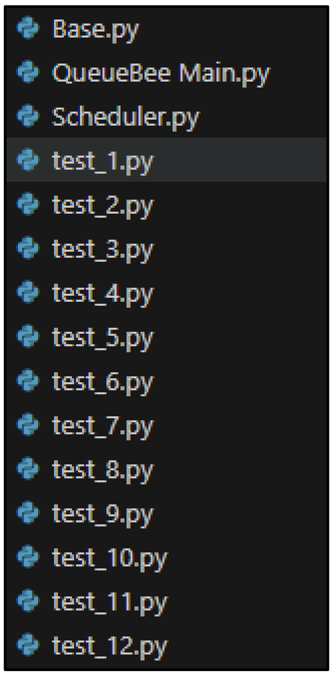
QueueBee	Develop a prototype model for the automation for software testing for the application.
	With the development of the prototype, ensure that the coding aspect is readable to handover to the client upon completing the prototype to further refine.
	Accuracy of reporting test case failure to enable the organization to resolve the back-end development.
Sunway University	PDPA form and ethical clearance form have to be submitted to the university for clearance.
	Incorporate concept & tools from Business analytics to the project to reflect the comprehension of the Business Analytical course.

Table 4: Initial requirement from stakeholder

3.3.3 Step 2: Prototype for Automation Software Testing

In creating the initial prototype, the development of the software system would be written in Python language and the web application automation will be facilitated by utilizing a python library called Selenium. To combat the complexity of coding while initiating the uniqueness of this research, Generative AI was introduced in one test case. Since the automation software would be handed over to QueueBee for actual application, it is vital to ensure all file and coding structures adhere to the industry standards.

- File structure



```
Base.py
QueueBee Main.py
Scheduler.py
test_1.py
test_2.py
test_3.py
test_4.py
test_5.py
test_6.py
test_7.py
test_8.py
test_9.py
test_10.py
test_11.py
test_12.py
```

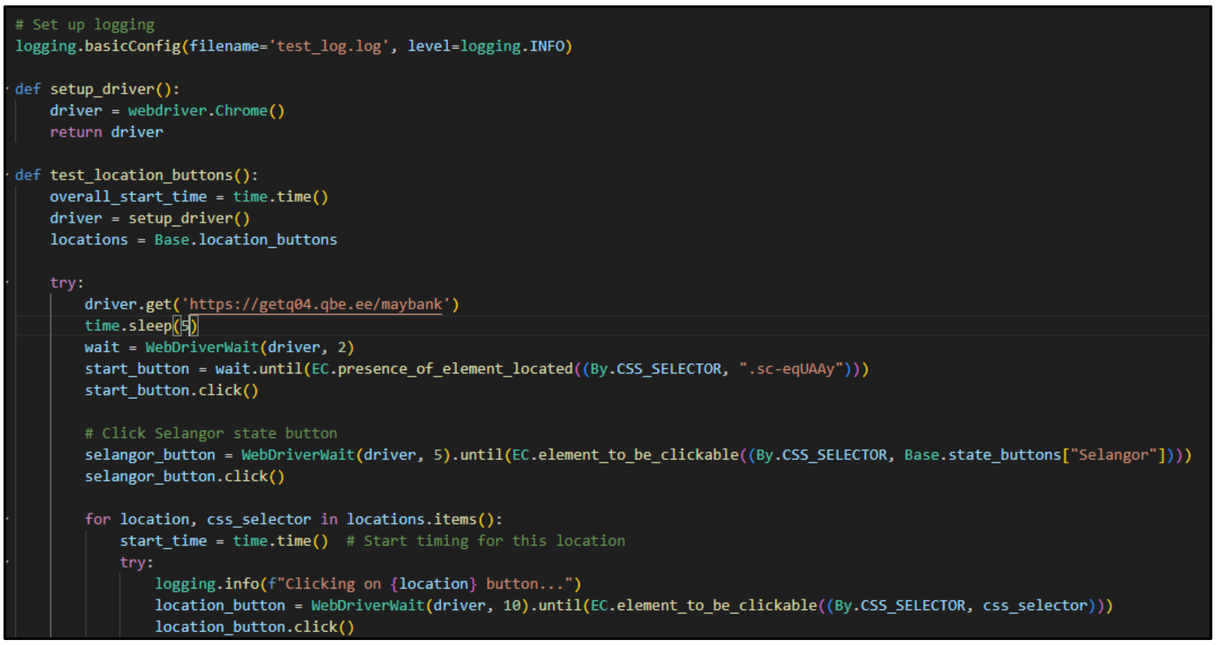
Figure 13: File structure in Visual Studio

The entire program consists of 4 main file structures, which are Base, QueueBee Main, Scheduler, and test_X. Table 5 below describes them in detail:

File structure	**Description**
Base.py	Consists of all the CSS or link paths to all testing buttons in a single screen.
QueueBee Main.py	Consists of all individual test script functions for calling and executing the respective test cases.
Schedule.py	Consists of the date and time for scheduling automated software testing.
test_X.py	Consists of detailed instructions for each test case.

Table 5: File structure for the automation software testing program

- Coding structure



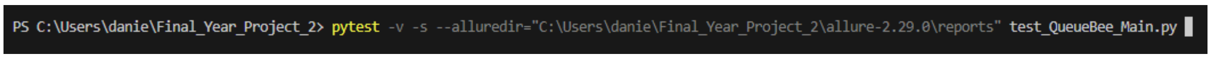
```
# Set up logging
logging.basicConfig(filename='test_log.log', level=logging.INFO)

def setup_driver():
    driver = webdriver.Chrome()
    return driver

def test_location_buttons():
    overall_start_time = time.time()
    driver = setup_driver()
    locations = Base.location_buttons

    try:
        driver.get('https://getq04.qbe.ee/maybank')
        time.sleep(5)
        wait = WebDriverWait(driver, 2)
        start_button = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, ".sc-eqUAAy")))
        start_button.click()

        # Click Selangor state button
        selangor_button = WebDriverWait(driver, 5).until(EC.element_to_be_clickable((By.CSS_SELECTOR, Base.state_buttons["Selangor"])))
        selangor_button.click()

        for location, css_selector in locations.items():
            start_time = time.time()  # Start timing for this location
            try:
                logging.info(f"Clicking on {location} button...")
                location_button = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, css_selector)))
                location_button.click()
```

Figure 14: Screenshot of codes for test case No.3

The above Figure 14 highlights the importance of a proper and standard coding structure. Firstly, each section should consist of 1 comment to briefly describe the usage of the function, while the comment must be short and concise. Secondly, all codes must be written in the most optimized way to avoid redundant instructions.

- Report generation

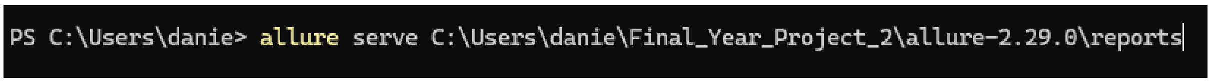
To further streamline the reporting process, this research adopted an automated report generation method with the usage of PyTest and Allure. As seen in Figure 15 and 16, within 2 lines of code, the system could generate a comprehensive report regarding all test cases in a few seconds.



```
PS C:\Users\danie\Final_Year_Project_2> pytest -v -s --alluredir="C:\Users\danie\Final_Year_Project_2\allure-2.29.0\reports" test_QueueBee_Main.py
```

Figure 15: Usage of Pytest in Visual Studio console

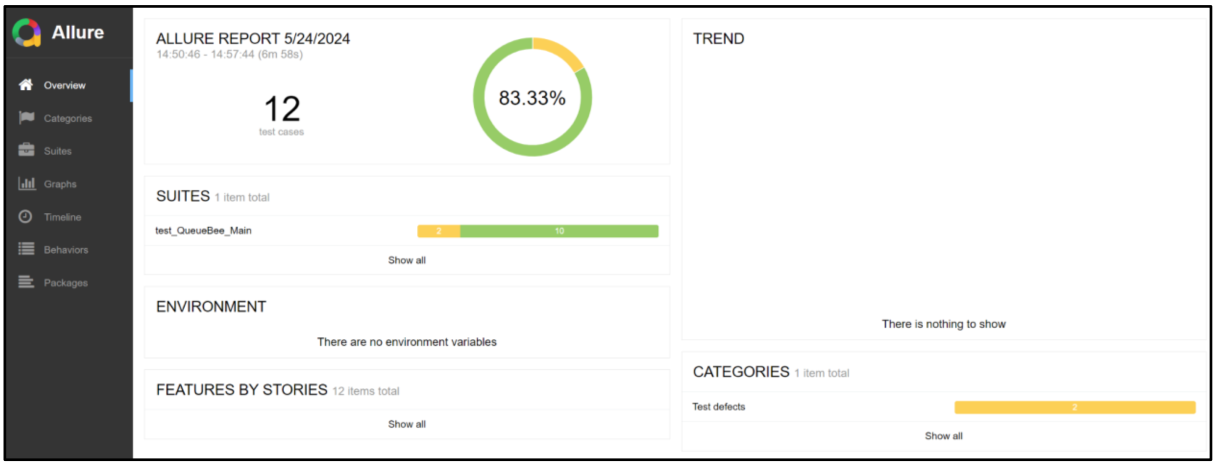
The purpose of adding the line above in the console is to capture all data and store in a file with a given directory.



```
PS C:\Users\danie> allure serve C:\Users\danie\Final_Year_Project_2\allure-2.29.0\reports
```

Figure 16: Usage of Allure in Command Prompt

The purpose of adding the line above in the command prompt is to activate and visualize the HTML code that stores the report data into a webpage on any default browser. Figure 17 below shows the main visual produced by Allure Reports.



ALLURE REPORT 5/24/2024
14:50:46 - 14:57:44 (6m 58s)
12 test cases
83.33%

TREND
There is nothing to show

SUITES 1 item total
test_QueueBee_Main
Show all

ENVIRONMENT
There are no environment variables

CATEGORIES 1 item total
Test defects
Show all

FEATURES BY STORIES 12 items total
Show all

Figure 17: Automatic report generation using Allure Reports

- Unique Element 1: Implementation of Generative AI

With the aim of developing a software automation system with the purpose of enhancing productivity and efficiency, artificial intelligence serves as a crucial component in reducing the amount of burden of executing repetitive tasks handled by employees. This current study introduces Generative Artificial Intelligence (GenAI) as a unique feature by introducing Image Recognition. In particular, the current study requires an Image Recognition algorithm for automating test cases, especially in color detection. By integrating GenAI, the process is efficient, reducing manual labor and allowing employees to concentrate on meaningful tasks. Meanwhile, instead of modeling a customer Image Recognition algorithm, this study utilizes the globally available Generative Pretrained Transformer (GPT) to perform such a scenario, which is using Google's Gemini.

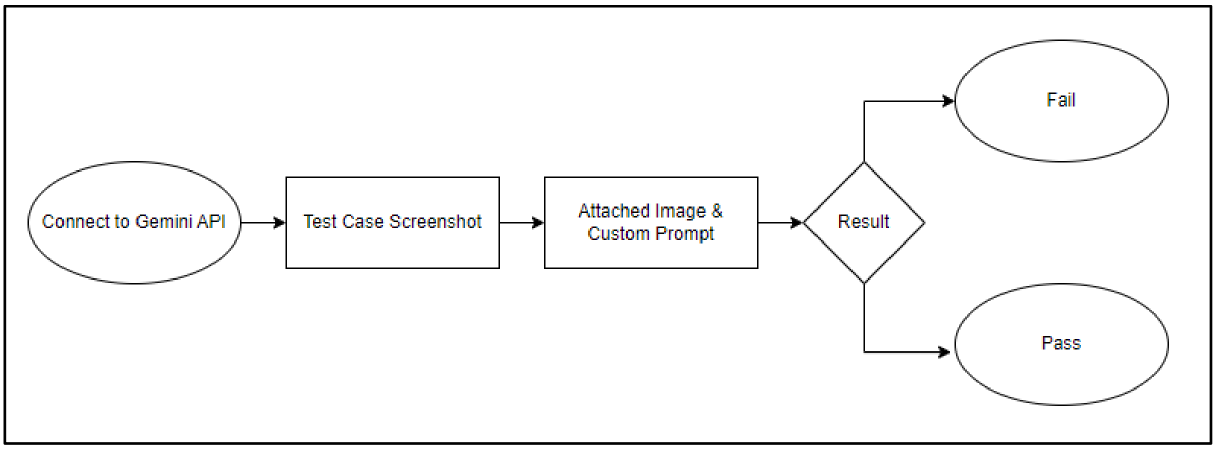
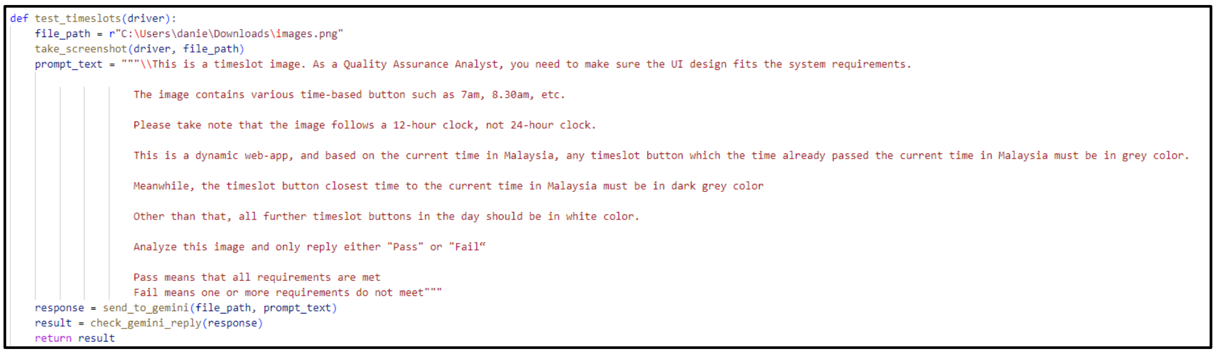


Figure 18: Flowchart for integrating Gen AI

Based on Figure 18, the API will be first connected to the code. Certain test scripts executed by the automation software will take a screenshot of the web page and it will be uploaded into Gemini with the following image and a pre-written prompt. The prompt was reviewed and updated multiple rounds to ensure that the response was compatible, resulting in a pass or a fail.



```python
def test_timeslots(driver):
    file_path = r"C:\Users\danie\Downloads\images.png"
    take_screenshot(driver, file_path)
    prompt_text = """\\This is a timeslot image. As a Quality Assurance Analyst, you need to make sure the UI design fits the system requirements.

                The image contains various time-based button such as 7am, 8.30am, etc.

                Please take note that the image follows a 12-hour clock, not 24-hour clock.

                This is a dynamic web-app, and based on the current time in Malaysia, any timeslot button which the time already passed the current time in Malaysia must be in grey color.

                Meanwhile, the timeslot button closest time to the current time in Malaysia must be in dark grey color

                Other than that, all further timeslot buttons in the day should be in white color.

                Analyze this image and only reply either "Pass" or "Fail"

                Pass means that all requirements are met
                Fail means one or more requirements do not meet"""
    response = send_to_gemini(file_path, prompt_text)
    result = check_gemini_reply(response)
    return result
```

Figure 19: Prompt used for Generative AI

Figure 19 above showcases the exact prompt used for Test 7, which the main goal is to check if the buttons on the User Interface are represented with the correct color choice. If Gemini returns as a failed test case, it means some buttons have incorrect color, which leads to error in the test case.

- Unique Element 2: Automated messaging feature using Twilio API

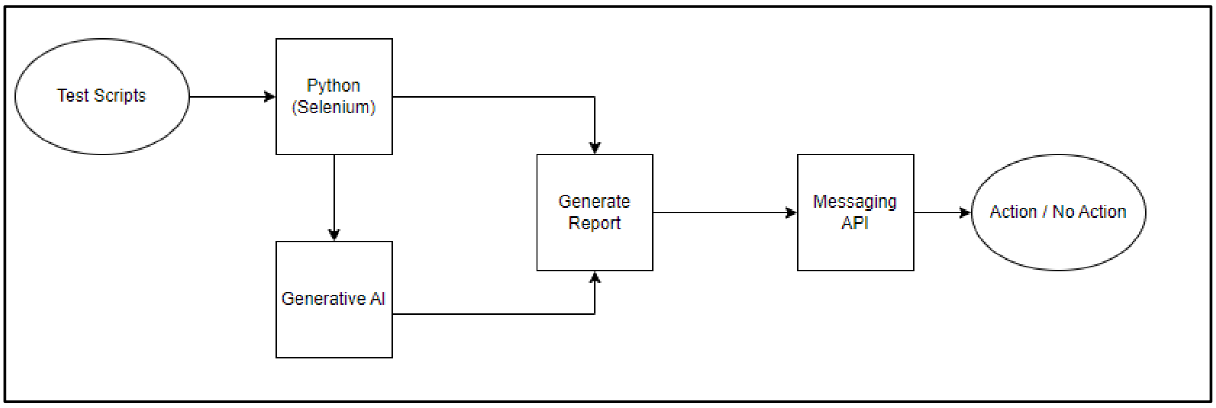
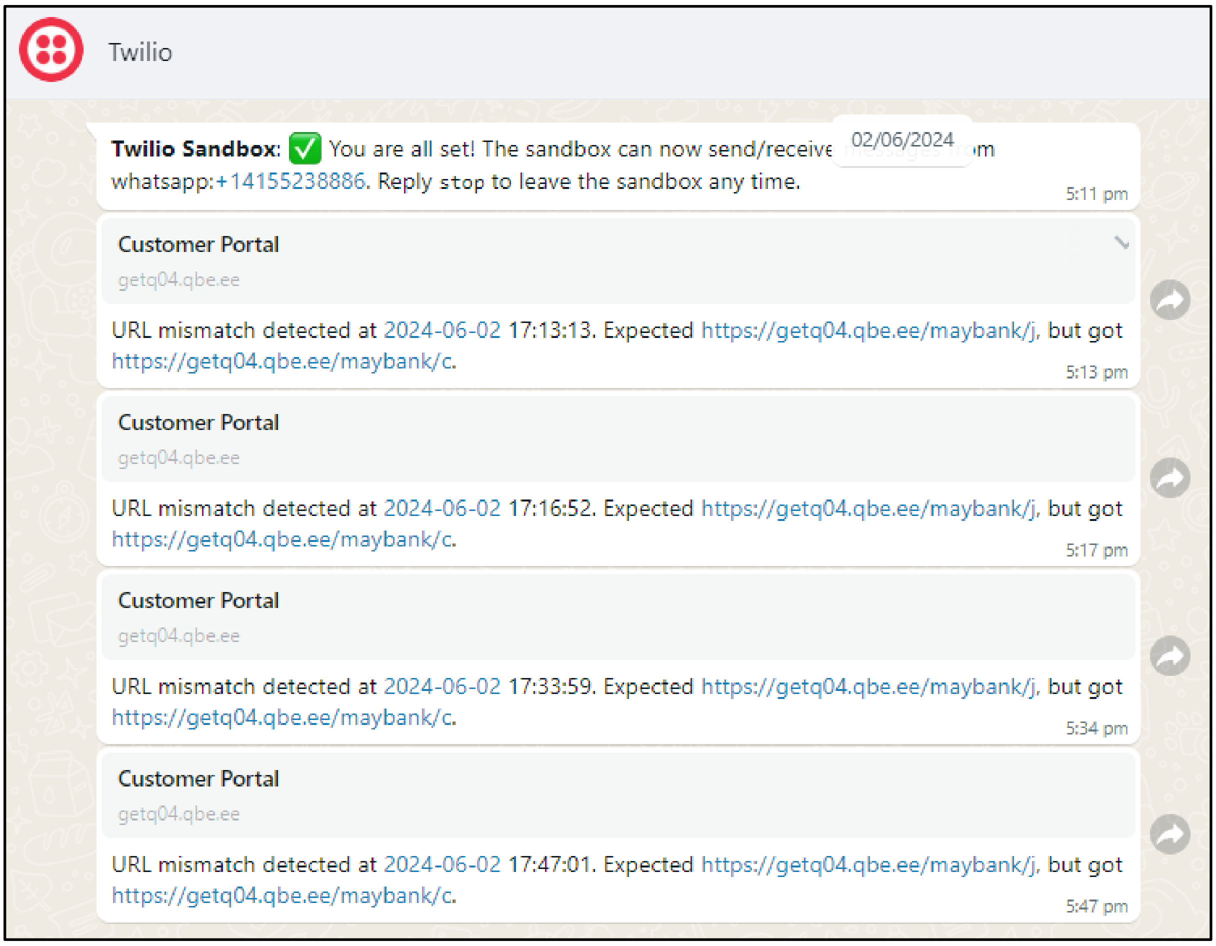


Figure 200: Flowchart for Auto-Generated Report

Figure 20 shows the overall flow for the generation of reports by the test scripts developed by the current research. By integrating Twilio's automated messaging API, it enables continuous monitoring throughout the day. It provides instant updates far quicker than manual notifications. In a traditional environment, the monitoring and execution of test cases are executed manually, only then the employee will generate a report with action to follow up. If an issue appears, it typically requires a physical person to manually notify the information to the relevant parties (direct manager). However, with the introduction of the SMS API, the process has shifted towards immediate notification when an error is detected. This approach optimizes the resource allocated within the department, significantly improving efficiency and response times. Figure 20 showcases the screenshot of how Twilio delivers an instant message on Whatsapp when a failed test case is detected.



Twilio

Twilio Sandbox: ✅ You are all set! The sandbox can now send/receive 02/06/2024 m whatsapp:+14155238886. Reply stop to leave the sandbox any time. 5:11 pm

Customer Portal
getq04.qbe.ee

URL mismatch detected at 2024-06-02 17:13:13. Expected https://getq04.qbe.ee/maybank/j, but got https://getq04.qbe.ee/maybank/c. 5:13 pm

Customer Portal
getq04.qbe.ee

URL mismatch detected at 2024-06-02 17:16:52. Expected https://getq04.qbe.ee/maybank/j, but got https://getq04.qbe.ee/maybank/c. 5:17 pm

Customer Portal
getq04.qbe.ee

URL mismatch detected at 2024-06-02 17:33:59. Expected https://getq04.qbe.ee/maybank/j, but got https://getq04.qbe.ee/maybank/c. 5:34 pm

Customer Portal
getq04.qbe.ee

URL mismatch detected at 2024-06-02 17:47:01. Expected https://getq04.qbe.ee/maybank/j, but got https://getq04.qbe.ee/maybank/c. 5:47 pm

Figure 211: Automated messaging feature on Whatsapp using Twilio

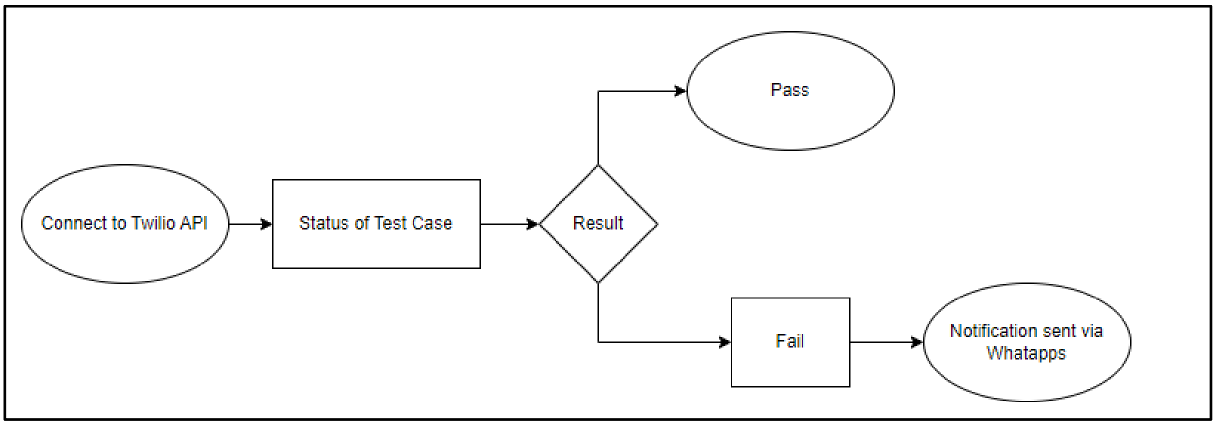
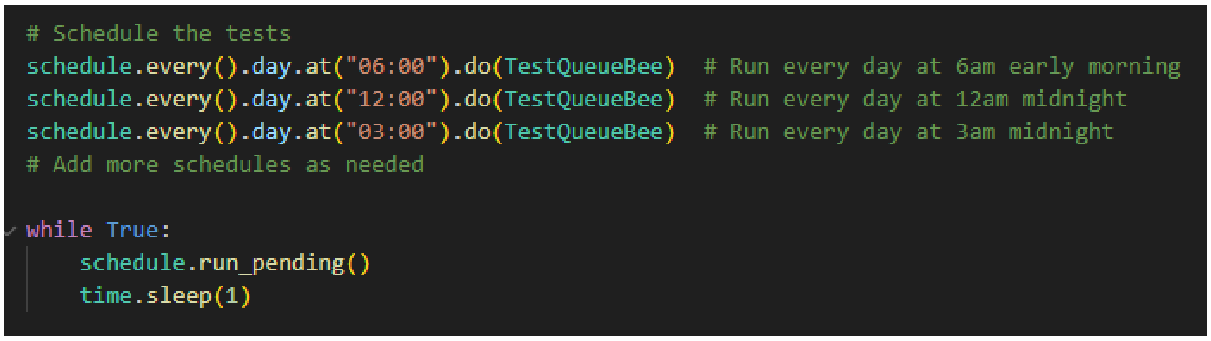


Figure 222: Flowchart for SMS API

Figure 22 provides more details regarding the automation process connected to Twilio’s API. When referreing back to Figure 20. the automation software is initially connected to Twilio’s API and would execute the test case, if the test case resulted in a failure, it would trigger the code and send a custom notification with the specification of the failure of the test case to the selected receiver via the SMS API. Else, the selected receiver will still receive a notification indicating that the automated software testing executed without any error. In doing so, this introduces the organization in leveraging the software to ensure continuous monitoring and provide immediate instantaneous notification to real-time critical emergencies, ensuring quick action.

- Unique element 3: Automatic scheduling system

With the current code, the automated software will execute testing at three specific times: 12 a.m., 3 a.m., and 6 a.m as shown below. The scheduling feature enhances productivity, as manually testing software can be time consuming, taking up to a day to complete for a single system. However, by introducing automation, efficiency is dramatically increased, it can perform continuous testing without fatigue and is less prone to human error. Additionally, in a traditional setting, testing might occur only during the development phase or if a bug appears. However, clients can now configure multiple intervals per day within the code to ensure the application operates as intended continuousl as seen in Figure 23 below.



```
# Schedule the tests
schedule.every().day.at("06:00").do(TestQueueBee)  # Run every day at 6am early morning
schedule.every().day.at("12:00").do(TestQueueBee)  # Run every day at 12am midnight
schedule.every().day.at("03:00").do(TestQueueBee)  # Run every day at 3am midnight
# Add more schedules as needed

while True:
    schedule.run_pending()
    time.sleep(1)
```

Figure 233: Codes for automated scheduling system in Visual Studio

3.3.4 Step 3: Stakeholder Evaluation

On 29 August 2024, the group will present the current study to the stakeholder. It will then be evaluated on the performance of the prototype. To gather insights regarding the automated software system, a brief survey will be conducted with the stakeholders. The feedback will be collected through a short survey, focusing on quantitative questions on selected stakeholders to determine the software system's performance. The feedback will be documented to ensure proper handover to the client to enable continuation of the development of the automated software system.

3.3.5 Step 4: Iterative refinement

Currently, this research did not facilitate any iterative requirement due to the fact that the prototype developed. The current study is currently on the first round of evaluation, therefore, the team will present the prototype to the client to get feedback to evaluate the performance of the software system against the initial requirement. With the end of the capstone project, the SDLC will conclude on the first cycle of the iteration therefore the project will be handed over to the client for continuation of the software system. Only if the organization engages the team externally, further discussion could be arranged.

3.4 Evaluation of automated and manual testing

The purpose of developing the automation software would be to improve efficiency without incurring additional cost as the company scales larger. Khankhoje (2023) paper mentioned that the purpose of automation was to streamline the process of early bugs detection within the queue management system application. During the initial requirement, the client had informed that the application had an exponential growth which resulted with over 3000 test scripts to be executed manually. With limited resources, limited test cases were carried out by the team which resulted with unforeseen bugs as the SDLC continued. Therefore, the researcher was to develop an automation software which is to be carried out automatically to improve efficiency. With the development of the automation test scripts, Khankhoje (2023) highlighted that it would require an initial investment and time, requiring allocation of resources towards code development instead of manually testing. Therefore, Khankhoje (2023) introduced the term Return Of Investment (ROI) as a method to evaluate the result between manual and automation testing.

3.4.1 Selection of Key Metrics

In the initial phase, the metric was based on Khankhoje (2023) key components of ROI in Test Automation. However, several of the key components were not applicable in this research as these components are not aligned with the current scope of the research. Only selective key components were extracted from the paper, as outlined in Table 6.

Key Components	**Definition**
Time Savings	The execution of test automation performs faster than when compared to manual testing.
Consistency	Ensure consistent results, ensuring reliability.

Table 6: Key components that were extracted from Khankhoje (2023) research

The key components shown above were the foundation of developing the following key metric to evaluate the automation system, as shown in Table 7 below.

Key Metric	**Description**
Execution Speed	Speed execution was derived from the time saving key components (Davrondzhon & Arne, 2020). Moreover, Davrondzhon & Arne (2020) study revealed that automation has improved the execution speed on a task by 50 times. Therefore, the current research would be evaluating the speed of executing the test scripts, measuring the speed differences.
Testing Accuracy	Testing accuracy was derived on the aspect of the software reliability. Despite not being included in the key component of Khankhoje (2023) research. The current study included this key metric to evaluate the software reliability to test the code robust against manual testing.
Productivity	Despite not listed within the key component, productivity plays a crucial role towards the current study in evaluating the ROI. Khankhoje (2023) paper reveals that resource allocation was a key component in understanding the ROI for test automation. With the client expanding exponentially, effective resource allocation plays a crucial role towards efficiency and productivity. Therefore, the current study would include the difference in terms of productivity between manual and automated software.
Scalability	Scalability was derived from the test coverage aspect, instead of focusing on the fixation of covering a wide range of scenarios. The current study shifted the focus towards scalability. The current study understood that the client was expanding its reach towards southeast asia (SEA) hence, the current metric would fit the current study narrative, adjusting to the client initial requirement (Khankhoje, 2023; Davrondzhon & Arne, 2020).
Consistency	To test the automation software reliability, consistency is an alternative metric of accuracy. Despite the confusion in similarity, the nuance between both is significant. Consistency focuses on the software replicability on the same set of tests, diminishing the probability of human error (Khankhoje, 2023).

Table 7: Key metrics used in the current study

3.4.2 Data collection method

There were two types of data collection to facillitate both manual and automated software testing.

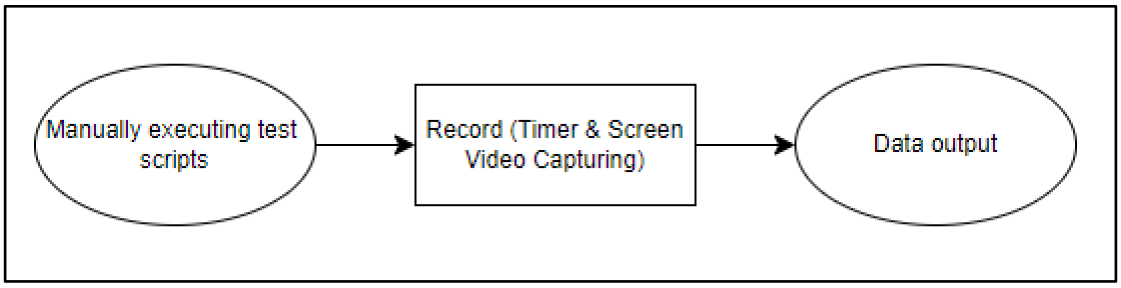

Manually executing test scripts
Record (Timer & Screen Video Capturing)
Data output

Figure 24: Flowchart for Data Collection (Manually)

As shown above in Figure 24, for manual testing, the current study implemented a traditional method for data collection. Each test script is recorded using screen capture and a timer. The results are then processed and presented in the results section.

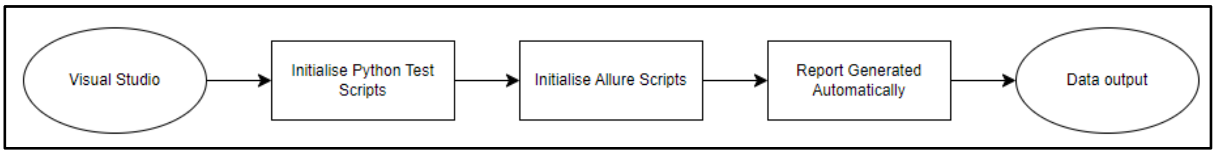

Visual Studio
Initialise Python Test Scripts
Initialise Allure Scripts
Report Generated Automatically
Data output

Figure 25: Flowchart for Data Collection (Automated)

As shown above in Figure 25, the automated software system utilizes an Integrated Development Environment (IDE) such as Visual Studio to develop the code. The system is coded using Python. The Allure module is then initialized and it will automatically generate the report, including metrics such as time taken for each test case. The data is collected and presented in the results section.

3.4.3 Data analysis and comparison method

- Execution speed

The results between the time taken for each test case using manual testing and automated testing were recorded and visualized using bar charts. The goal was to observe whether there's a significant difference for the time taken to complete each test case between manual and automation.

- Test accuracy

The experiment was conducted five times to show the difference between manual testing and automated testing. The main goal was to observe whether each testing would result in different accuracy. Below shows the equation to calculate testing accuracy, which would be repeated five times for both manual and automation testing.

$$\frac{\gamma}{n} \times 100\% = Test\ accuracy\ (\%)$$

$$where...$$

$$\gamma = Number\ of\ passed\ test\ cases$$

$$n = Total\ test\ cases$$

- Consistency

The experiment was conducted three times in three different environments, which are quiet, noisy, and multi-tasking as outlined in Table 8. The main goal was to observe whether each testing would result in different testing duration depending on different environments.

Quiet	The experiment was conducted in an enclosed meeting room without any external disturbance.
Noisy	The experiment was conducted in an open area with various noise disturbances such as crowd talking, walking, and shouting.
Multi-tasking	The experiment was conducted under the simulation of a real-life workplace where the tester was required to perform other minor, trivial or random tasks in the midst of performing each test case.

Table 8: Definition of environment for testing

The results were visualized in a grouped bar chart to showcase the total duration taken to complete each test case between manual and automation.

- Productivity

The experiment began by first calculating the average speed per test case between manual and automation. Below showcases the equation used to calculate the average value.

$$\text{Average speed per test cases (seconds)} = \frac{\delta}{n} \times 100\%$$

$$where\ldots$$

$$\delta = Total\ duration\ for\ all\ test\ cases\ (seconds)$$

$$n = Total\ test\ cases$$

A simple simulation was conducted by calculating the average time per test case for both automation and manual testing, and then multiplying this average time by the number of test cases to estimate the total time required for 1,000 test cases. The main goal is to observe the simulated time taken from 1 to 1,000 test cases between automation and manual. Below showcases the equations used to calculate the realistic time taken between automation and manual.

$$Time\ taken\ for\ automation\ (seconds)\ =\ \mu\ \times\ n$$

$$where...$$
$$\mu\ =\ Average\ speed\ per\ test\ case\ (seconds)$$
$$n\ =\ Number\ of\ test\ cases\ (1\ to\ 1{,}000)$$

$$Time\ taken\ for\ manual\ (seconds)\ =\ (\mu\ \times\ n)\ +\ \alpha\ +\ \beta$$

$$where...$$
$$\alpha\ = Constant\ of\ 3000\ seconds\ human\ rest\ per\ 10\ test\ cases$$
$$\beta\ = Constant\ of\ 57600\ seconds, if\ [(\mu\ \times\ n)\ +\ \alpha\ ]\ > 28{,}800\ seconds$$
$$Because\ 28{,}800\ seconds\ reflect\ the\ standard\ 8\ working\ hours$$
$$of\ 9am\ to\ 5pm$$
$$Meanwhile, the\ 57600\ seconds\ is\ the\ remaining\ 16\ hours\ of$$
$$non\ working\ time$$

- Scalability

The experiment simulated the time required for both automation and manual testing based on the number of applications to be tested. The benchmark scenario involved 600 test cases per application, with the experiment ranging from 1 to 10 applications. From previous calculations, 600 test cases would take 20,820 seconds with automation and 108,750 seconds with manual testing. The simulation was performed by multiplying the time per application by the number of applications. The main goal was to observe how automation can test all applications in parallel, while manual testing is restricted to sequential execution. Below showcases the equation used to calculate the time taken between automation and manual.

3.4.4 Business analytics

After conducting a thorough analysis of all key metrics, this section offers a high-level overview of the automation system using visuals such as graphs and bar charts created in Power BI. The key areas of discussion include insights visualization, comparative analysis, and strategic decision-making, as seen in Table 9 below.

Key Insights Visualization	This part presents a clear and concise representation of the key findings using visual aids. The focus is on highlighting trends and patterns that provide a deeper understanding of the system's performance.
Comparative Analysis	This section compares automation and manual testing across different metrics, using side-by-side visual comparisons to showcase differences in performance, speed, and efficiency.
Strategic Decision Making	This section helps stakeholders understand how to optimize testing strategies moving forward.

Table 9: Business Analytics component

3.5 Readiness for Business Change Process (BCP)

Within this section, the current study aims to assess the organization's current state transition into a desired future state. Before implementing BCP into the organization, the current study would be conducting a survey to assess the readiness of the company towards the change. In doing so, it would reflect if the current employee within the organization exhibits any potential sign of resistance towards change. This data would be crucial towards the next stage of addressing the resistance during th BCP (Meniberu, 2015). Currently, the study does not address and it could be addressed in future studies. As shown in Figure 26, the current process would be to design the questionnaire, it would be then sent to the software department within the organization. The current would then conduct an analysis and discuss the finding.

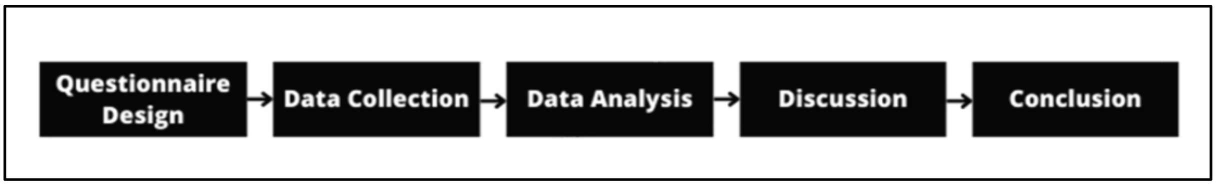


Figure 26: Research methodological workflow for Objective 3

3.5.1 Questionnaire design

A 5-point Likert Scale was implemented as the methodology for collecting quantitative questionnaire data to assess the readiness of the company in implementing the automated software testing system (Alamansoori, Rahman & Memon, 2021). The 5-point Likert scale ranges from a scale of 1 (related to a negative outcome) to 5 (related to a positive outcome) in terms of agreement. Table 10 below outlines the meaning of each scale:

Scale	**Meaning**
1	Strongly disagree
2	Disagree
3	Neutral
4	Agree
5	Strongly agree

Table 10: Meaning of each point in 5-point Likert Scale

The survey of this research referenced the questionnaires conducted by Mukred et al (2019) that contained 7 sections related to business process change, which are top management support, cultural readiness, learning capacity, internal training, documentation, policy, and financial. However, within the current study, only five of the factors were selected and implemented which were mentioned in Table 11. With the deployment of the automation software, the current study would have to evaluate the readiness of the organization towards introducing new software into the work space. Based on Lagrosen et al (2011) research paper, the survey criteria for BPC are shown below in Table 11.

BPC Criteria	Description
Top management support	With the introduction of software automation, accessing the top management opinion towards the automation software weighs heavily. The opinion reflects the level of initiative that will be placed towards introducing the process to improve the business process. Therefore, in the perspective of the top management, if the new process does not create value. The management may not provide initiative towards the business process. The lack of initiative would reflect throughout the lower level of the hierarchy.
Internal Training	With the understanding that the organization could be potentially fresh towards software automation testing. It would be crucial to organize an internal training for the organization to elaborate the automation software concept. The target audience of the training would be towards the middle management. This would be due to middle management having the influences to address issues with the new software and cultivating the organization readiness culture.
Documentation	Documentation would be essentially to assure the organization readiness. The documentation will be transferred to the organization to enable that the team can refer in case of encountering any issue.
Policy	To assure the organization adheres to industry standard policies to protect client’s data and information when performing the automation tests on their respective application.
Financial support	To assure the organization has enough capital investment to adopt an automation software testing system and build a team of developers to code the program.

Table 11: Criteria for Business Process Change

In total, the questionnaire consists of 25 questions, with 5 questions for each section of Business Process Change (BPC) mentioned above. All questions are presented in the Appendix. Table 12 below outlines the structure of the survey.

Section	Number of Questions
Top management support	5
Internal training	5
Documentation	5
Policy	5
Financial support	5

Table 12: Sections contained in the survey

3.5.2 Sampling method

The main participants in this survey were the top management and main engineering team in QueueBee, which consists of 15 employees. As a result, a purposive sampling method was used to select desired participants. Purposive sampling is a non-probability sampling method where researchers use their discretion to select participants from the general population for their surveys (Alchemer, 2021). This sampling method was selected in this study to ensure all participants are either the employees involved with software testing or stakeholders of the company, without including any participants that were not directly involved in the software testing process. Thus, this method would maximize the accuracy when evaluating the readiness while eliminating potential outliers.

3.5.3 Data collection method

The research team conducted a physical presentation at QueueBee's office to showcase the implementation of automated software testing. All selected participants must be present during the presentation. After the session, all participants proceeded to answer all the questions in the given survey using Google Form. All respondents must click the "Submit" button to complete the survey.

3.5.4 Data analysis method

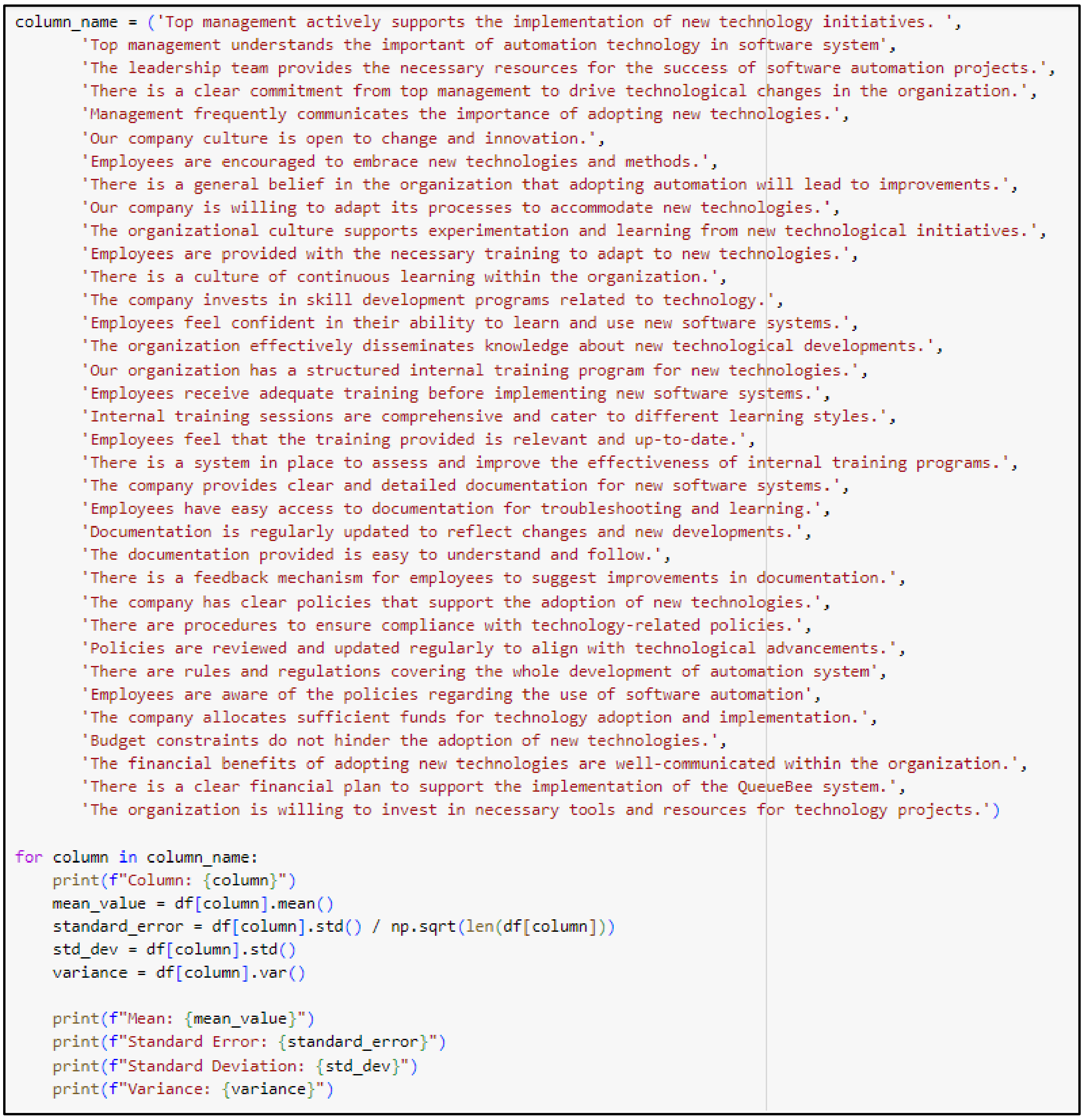
To ensure no data was missing before moving forward with the data analysis, two rounds of data verification were conducted. All data was retrieved from Google form (refer to Appendix A), downloaded in a "CSV" format, and use python for data analysis and visual representation.

A descriptive analysis was conducted to display and summarize the result of the survey (Rawat, 2021). Table 13 outlines the key metrics included in the anlaysis.

Key Metric	**Definition**
Mean	The mean is the average of all the values. It is calculated by calculating the total sum of all the values and dividing by the number of values.
Standard Error	It measures the accuracy of a sample mean that represents the population mean. The standard error will show the sample mean in comparison to the actual population. In doing so, it shows how accurately the sample mean represents the actual population.
Standard Deviation	It measures the dispersion within a set of values around the mean. A low standard deviation means that the values are close to the mean while a high standard deviation means that it is far away from the mean (more variability).
Variance	It measures the spread of the dataset by calculating the square differences between each value and the mean. Variance shows how diverse the values are in the dataset.

Table 13: Definition of key metric for survey

Since the third objective of this research was to determine the readiness of the company to implement the automated software testing system, the 5-point Likert scale was the best method for assessing the company's decision.



```
column_name = ('Top management actively supports the implementation of new technology initiatives. ',
       'Top management understands the important of automation technology in software system',
       'The leadership team provides the necessary resources for the success of software automation projects.',
       'There is a clear commitment from top management to drive technological changes in the organization.',
       'Management frequently communicates the importance of adopting new technologies.',
       'Our company culture is open to change and innovation.',
       'Employees are encouraged to embrace new technologies and methods.',
       'There is a general belief in the organization that adopting automation will lead to improvements.',
       'Our company is willing to adapt its processes to accommodate new technologies.',
       'The organizational culture supports experimentation and learning from new technological initiatives.',
       'Employees are provided with the necessary training to adapt to new technologies.',
       'There is a culture of continuous learning within the organization.',
       'The company invests in skill development programs related to technology.',
       'Employees feel confident in their ability to learn and use new software systems.',
       'The organization effectively disseminates knowledge about new technological developments.',
       'Our organization has a structured internal training program for new technologies.',
       'Employees receive adequate training before implementing new software systems.',
       'Internal training sessions are comprehensive and cater to different learning styles.',
       'Employees feel that the training provided is relevant and up-to-date.',
       'There is a system in place to assess and improve the effectiveness of internal training programs.',
       'The company provides clear and detailed documentation for new software systems.',
       'Employees have easy access to documentation for troubleshooting and learning.',
       'Documentation is regularly updated to reflect changes and new developments.',
       'The documentation provided is easy to understand and follow.',
       'There is a feedback mechanism for employees to suggest improvements in documentation.',
       'The company has clear policies that support the adoption of new technologies.',
       'There are procedures to ensure compliance with technology-related policies.',
       'Policies are reviewed and updated regularly to align with technological advancements.',
       'There are rules and regulations covering the whole development of automation system',
       'Employees are aware of the policies regarding the use of software automation',
       'The company allocates sufficient funds for technology adoption and implementation.',
       'Budget constraints do not hinder the adoption of new technologies.',
       'The financial benefits of adopting new technologies are well-communicated within the organization.',
       'There is a clear financial plan to support the implementation of the QueueBee system.',
       'The organization is willing to invest in necessary tools and resources for technology projects.')

for column in column_name:
    print(f"Column: {column}")
    mean_value = df[column].mean()
    standard_error = df[column].std() / np.sqrt(len(df[column]))
    std_dev = df[column].std()
    variance = df[column].var()

    print(f"Mean: {mean_value}")
    print(f"Standard Error: {standard_error}")
    print(f"Standard Deviation: {std_dev}")
    print(f"Variance: {variance}")
```

Figure 27: Python functions for calculating the key metric in Visual Studio

By calculating the key metrics of each section, the result would reveal the company’s level of readiness towards implementing the automation system. Figure 27 showcases the codes used to calculate the key metrics such as mean and standard deviation.

CHAPTER 4

FINDINGS & DISCUSSION

4.1 Chapter Overview

This section discusses the overall results collected after conducting the experiment. The results are observed, analyzed, and justified to achieve all three objectives mentioned in the introductory section of the report.

4.2 Development of Automation Testing Software

This section showcases the development of the automation testing software, including the User Interfaces (UIs), objective of each test case, and report analysis. Refer to Table 14 below:

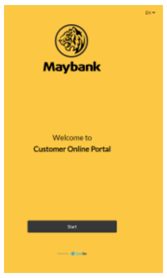
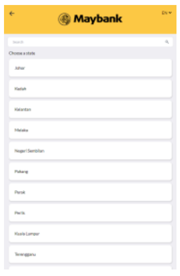
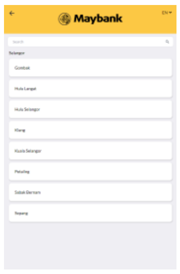
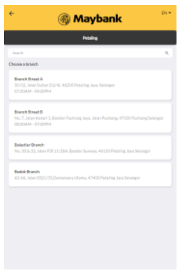
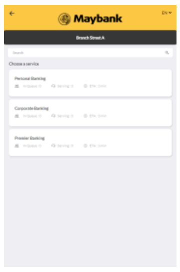
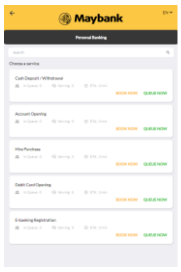
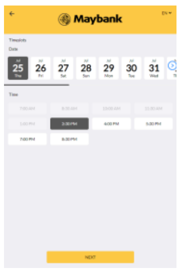
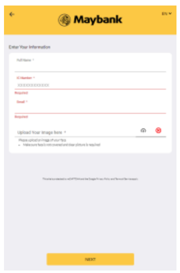
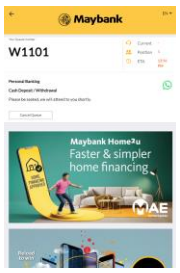
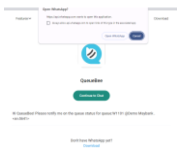
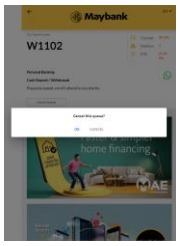
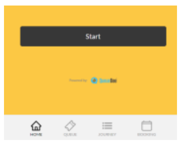
					
1: Home page	2: State	3: Location	4: Branch	5: Service	6: Sub-service
					
7: Book now	8: Queue now	9: Queue page	10: Whatsapp	11: Cancel	12: Menu bar

Table 14: Image of User Interface

The entire software testing process revolves around 12 user interfaces, which begins from the Home page until the Queueing page. This project's primary objective is to develop a software automation system that automatically runs various test cases revolving around these 12 user interfaces. Refer to Table 15 below.

Test Case	Objective	Flow
1	To test the start button	1
2	To test all state buttons are clickable and lead to the correct page and URL	1,2
3	To test all location buttons within state item are clickable	1,2,3
4	To test all branch buttons are clickable and lead to the correct page and URL	1,2,3,4
5	To test all service buttons are clickable and lead to to the correct page and URL	1,2,3,4,5
6	To test all sub-service buttons are clickable and lead to the correct page and URL	1,2,3,4,5,6
7	To test all date and time buttons at the timeslot page are clickable and lead to the correct page using Generative AI via Gemini API	1,2,3,4,5,6,7
8	To test the functionality of the form page by filling in random personal information	1,2,3,4,5,6,8
9	Test the functionality of the Whatsapp button at the viewing queue number page	1,2,3,4,5,6,8,9,10
10	Test the functionality of canceling a queue number after the queueing starts	1,2,3,4,5,6,8,9,11
11	To start a new queue while the current queue number is still active	1,2,3,4,5,6,8,9,12
12	To test the functionality of automated SMS to indicate a failed test case using Twilio API	1,2,3

Table 15: Test Cases for each interface

Table 15 above showcases the objective and flow of each test case. This information is treated as the main algorithm for programming each test case to fit the respective testing objective. Once the automation software testing is complete, a report will be automatically generated to present the overall result, containing numbers and charts to visually convey the testing result.

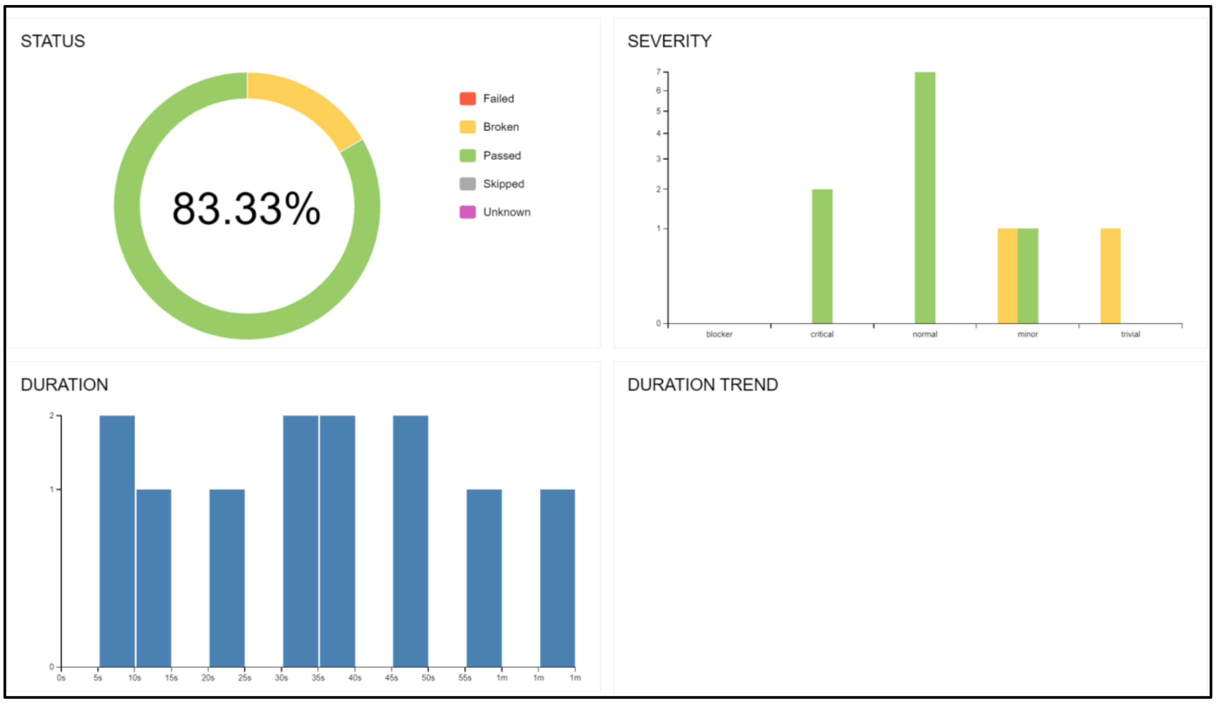


Figure 28: Report generated by Selenium

4.2.1 Status analysis

As seen on Figure 28, There are 5 status segmentation for the overall software automation testing, which are passed, broken, failed, skipped, and unknown:

- **Passed:** The majority of the tests, accounting for 83.33%, have passed. This indicates a relatively high success rate.
- **Broken:** There is a segment marked as "Broken", which suggests some tests encountered issues but didn't necessarily fail due to application bugs.
- **Failed:** There are some tests that failed, indicated by the red segment.
- **Skipped:** This segment is either very small or non-existent, indicating that most tests were executed without being skipped.
- **Unknown:** This segment is also very small or non-existent, suggesting that there were few or no tests with an unclear outcome.

4.2.2 Duration analysis

The duration chart shows that most test cases are executed within a time frame ranging from 5 seconds to 1 minute. The execution time is fairly distributed with a peak around the 5-10 seconds and 30-40 seconds marks.

4.2.3 Severity analysis

There are 5 levels of severity, which are blocker, critical, normal, minor, and trivial that relate to the level of importance of each test case. If a green color bar appears under critical, it means a very important test case has passed the test successfully. Conversely, if a yellow color bar appears under critical, it means a very important test case has failed and requires immediate mitigation.

Most of the issues detected are of normal severity, which implies that while there are issues, they are not critically affecting the application's functionality.

4.2.4 General comparison between automation and manual

Table 16 showcases a general comparison between automation testing system and manual testing system.

Elements	**Automation**	**Manual**
Automated documentation	Yes	No
Parallel testing	Yes	No
Generative AI	Yes	No
Automated scheduling system	Yes	No
Automated messaging feature	Yes	No
Automated report generation	Yes	No

Table 16: Comparison between automation testing and manual testing

Based on Table 16, it is apparent that automation offers more upside than manual since all elements presented in automation are not presented in manual testing. In terms of documentation, automation software testing can automatically record the time spent on each test case down to milliseconds (Swanepoel et al., 2010). This feature helps with better tracking and reporting on testing processes. Meanwhile, manual testing lacks automatic time recording, but rather relying on manual testers to record the time, leading to inconsistency and inaccuracy (Sajji, 2023).

In terms of parallel testing, it refers to the ability of performing multiple test cases simultaneously, which can only be achieved by automation (Shahin et al., 2017). This is because the automation could concurrently be run on multiple Visual Studio files without overlapping (Avritzer et al., 2022). This capability significantly reduces the overall testing time and scalability, which is further discussed in the “Scalability” section below. Meanwhile, under the assumption of one software tester available in the company, manual testing is sequential, with each test case executed one after the other since a human being could not perform two test cases at once. As a result, manual testing is too time-consuming, especially for large projects.

In terms of Generative AI (GenAI), automated testing can leverage GenAI to validate a test case using image recognition and prompt engineering, which could replace the usual hard coding method to program every single instruction (Layman, 2024). This enhances the testing process in both speed and efficiency for both testing and programming time. On the other hand, manual testing does not support the feature of GenAI, thus limiting the further upgrade in software testing.

In terms of an automated scheduling system, automated testing can be scheduled to run during odd hours, such as nights, weekends, and even public holidays. This allows continuous testing without human intervention (Cauwenberghe et al., 2012). On the flip side, manual testing requires actual human testers to conduct each test case, resulting in various challenges to perform testing during odd-hours and leading to poor flexibility.

In terms of automated messaging features, the automated system can send immediate notification to any desired individual to stay informed about the status of automated software testing while being away (Reuter et al., 2019; Yadav et al., 2021). Manual testing systems only provide manual communication, which can lead to delays and misleading information.

In terms of automated reporting, the automated software testing system can generate detailed reports automatically using Allure Reports. This feature provides a comprehensive dashboard showcasing all the test cases included, time taken, status, coverage, and more. Meanwhile, manual testing often leads to manual reporting, which may result in human errors, poor data collection, bad usage of words and ultimately, time-consuming (Wiklund et al., 2017).

4.2.5 Discussion Summary

Overall, the automated software testing system offers various advantages over manual testing while achieving the first objective of successfully developing an automation testing system. The next section discusses the experimentation between automation and manual testing in detail, covering metrics such as speed, accuracy, productivity, scalability, consistency, and cost.

4.2 Performance evaluation

This section showcases the experiment results of each key metrics between automation testing and manual testing, followed by its respective relevant discussion. The main goal of this section is to evaluate the performance between automation and manual testing, reflecting the second objective of this research.

4.2.1 Execution speed

Table 17 offers a comprehensive comparison of the time taken to execute each test case using automated testing versus manual testing.

- **Experiment result**

The results reveal significant differences in execution times between the two methods.

Test Case	Time taken for automation	Time taken for manual labor
1	9s 335ms	4s
2	39s 694ms	65s
3	14s 158ms	63s
4	8s 950ms	56s
5	24s 032ms	55s
6	56s 926ms	65s
7	65s	35s
8	35s 541ms	59s
9	46s 094ms	72s
10	34s 486ms	68s
11	33s 780ms	87s
12	48s 415ms	34s

Table 17: Comparison result of automation and manual testing

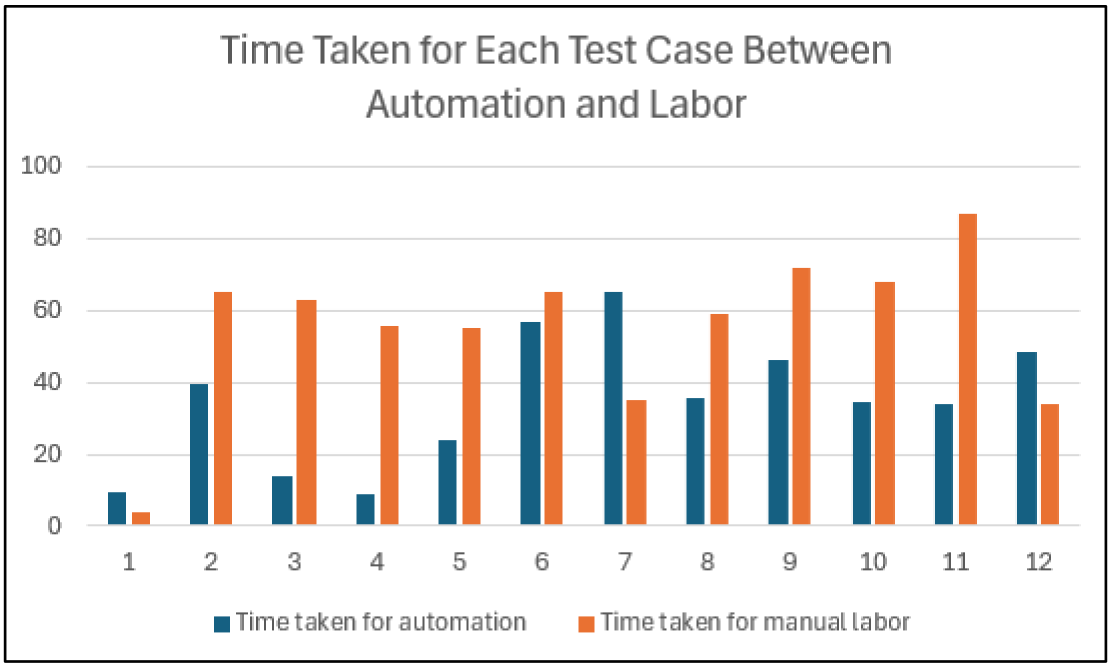


Figure 29: Bar chart for execution speed

- **Discussion**

Based on Figure 29, in most test cases, automated testing outperformed manual testing in terms of speed. For instance, in Test Case 2, automated testing took 39.694 seconds, whereas manual testing took 65 seconds. Similarly, in Test Case 4, automated testing was completed in 8.950 seconds compared to 56 seconds for manual testing. These results demonstrate a consistent trend where automated testing significantly reduces the time required to execute test cases.

Judging by the table above, only two cases (Task 7 and Task 12) where manual testing is faster than automation. For Task 7, manual testing is quicker than automation because the code utilized Generative AI (GenAI) which took time connecting to Gemini using its API. The purpose is to replace manual button testing and utilizes GenAI and prompt engineering to efficiently achieve the testing objective. This scenario would deem useful when testing a screen with a significant amount of buttons, which the automation system would ultimately surpass manual testing in terms of execution speed.

For Task 12, the main objective is to notify a person-in-charge when the test detects an error. In this case, manual testing is quicker than automation because the code utilizes Twilio's API to send a notification to a receiver via Whatsapp. Meanwhile, manual testing works by manualling messaging the person-in-charge when a software tester finds an error in the screen. However, having an automated SMS has a huge upside since a manual tester is not able to work overnight while the automation system could run tests at all times and even during odd hours without any supervision and immediately notifies when an error occurs.

4.2.2 Testing accuracy

To evaluate the accuracy of the automated and manual testing methods, the following procedure outlined in Table 18 was implemented.

Step 1: **Test execution**	Each test was repeated four times to ensure consistency and reliability of the results. This repetition helps identify any anomalies or inconsistencies in the testing process.
Step 2: **Manual review**	All manual testing sessions were recorded using a screen recording tool. The recorded sessions were then reviewed manually to ensure that all required steps were followed accurately. Any discrepancies or missed actions were noted and counted as a failure. Meanwhile, for automation testing, an automated report is generated which could be used for manual review.
Step 3: **Error identification**	In manual testing, errors such as forgetting to click a button or navigating to an incorrect URL were counted as failures. In automated testing, any script failures or incorrect outcomes were also counted as failures.

Table 18: Procedure for manual and automated testing (accuracy)

- **Experiment Results**

Table 19 summarizes the pass/fail outcomes for both manual and automated testing across different test cases and test sessions.

Test case	**Test 1**		**Test 2**		**Test 3**		**Test 4**		**Test 5**	
	Manual	Auto	Manual	Auto	Manual	Auto	Manual	Auto	Manual	Auto
1	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	**Fail**	Pass
3	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
4	Pass	Pass	Pass	Pass	Pass	Pass	**Fail**	Pass	**Fail**	Pass
5	Pass	Pass	**Fail**	Pass	Pass	Pass	**Fail**	Pass	Pass	Pass
6	Pass	Pass	**Fail**	Pass	Pass	Pass	**Fail**	Pass	**Fail**	Pass
7	Pass	Pass	Pass	Pass	**Fail**	Pass	Pass	Pass	**Fail**	Pass
8	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
9	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
10	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
11	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
12	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

Table 19: Binary result for the comparison of manual and automated testing.

The overall accuracy for manual and automated testing sessions are presented below in Table 20.

Test sessions	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**
Manual	100%	83%	92%	75%	67%
Auto	100%	100%	100%	100%	100%

Table 20: Result for the comparison of manual and automated testing in percentage

From the data collected, it is evident that automated testing consistently achieves a 100% accuracy across all test cases and sessions. This contrasts with manual testing, where the accuracy fluctuates, with the lowest being 67% in Test 5 and the highest being 100% in Test 1. Furthermore, the accuracy of manual testing gradually fell from 100% down to 67% at the 5th testing round.

- **Discussions**

This section discusses four key observations regarding the chart above, which are consistency, human error, efficiency, and reliability.

In terms of consistency, automated testing provided consistent results across all five experiments, unlike manual testing which accounted for several failed test cases. This consistency is critical in ensuring reliability and repeatability of the tests, reducing the likelihood of human error (Altmann et al., 2019).

In terms of human error, manual testing is prone to human errors, such as forgetting to test certain buttons or making mistakes in the test execution process. These errors contributed to the lower pass rates observed in manual testing sessions (Altmann et al., 2019).

In terms of efficiency, automated testing was more efficient in terms of time and resource utilization. Once the scripts were written and validated, they could be run multiple times without additional effort, unlike manual testing which required repeated human intervention (Kazimov et al., 2021).

In terms of reliability, automated tests are higher due to their ability to precisely follow predefined steps without deviation. In contrast, manual testing's reliability is impacted by the tester's attention to detail and consistency (Chandrasekaran* et al., 2019).

4.2.3 Productivity

To evaluate the productivity between automated and manual testing methods, the total duration for testing 10 cases was taken and multiplied by 10 folds, resulting in the estimated total duration up to 100 test cases. The goal is to perform an ideal and a realistic comparison between manual and automation testing in relation to the number of test cases (Garousi & Pfahl, 2015).

- **Experiment Results**

Table 21 below summarizes the actual time taken for both manual and automated testing along with its percentage difference.

	Automation (seconds)	**Manual (seconds)**	**Percentage difference**
Total duration for all test cases (seconds)	416	663	59.37%
Average speed per test case (seconds)	34.7	55.25	59.25%

Table 21: Difference of automation and manual in seconds

Table 22 below outlines the estimated time taken for both manual and automated testing across different sets of test cases, ranging from 10 to 1000 test cases.

Number of test cases	**Automation (seconds)**	**Manual (seconds)**
1	34.7	55.25
2	69.4	110.5
3	104.1	165.75
4	138.8	221
5	173.5	276.25
…	…	…
100	3470	5525

Table 22: Estimated duration as the number of test cases increases.

- **Discussions**

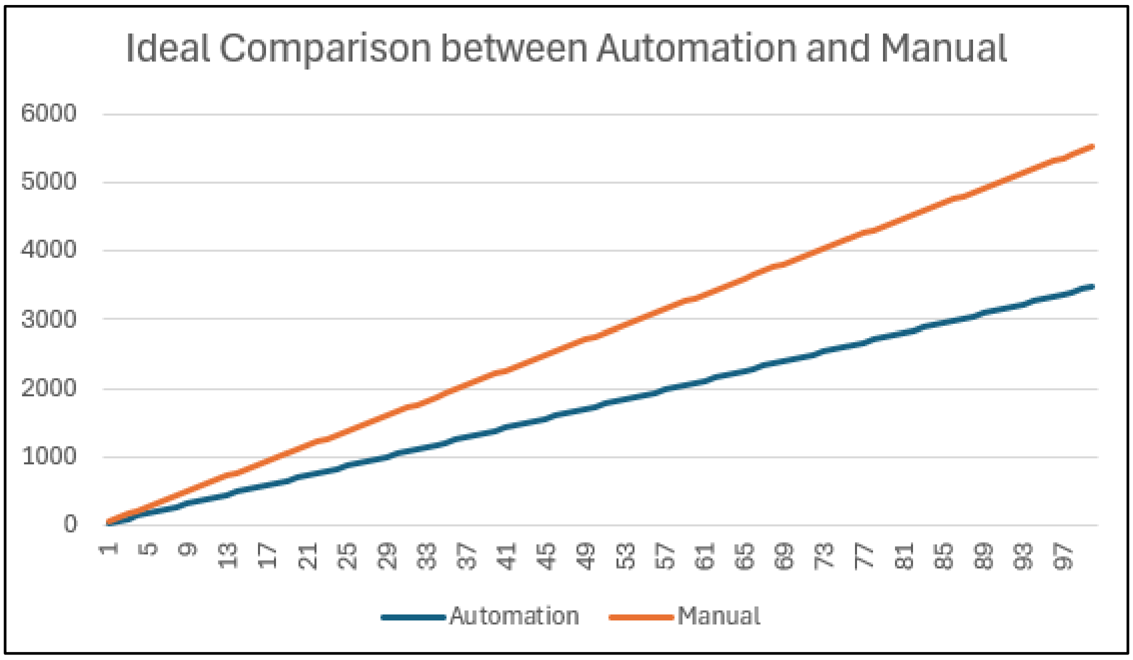


Figure 30: Ideal comparison between automation and manual

Figure 30 above indicates a linear relationship between the number of test cases and the time taken for both manual and automated testing. As the number of test cases increases, the time required for both methods increases proportionally. However, manual testing always takes more time than automation. The chart clearly demonstrates that automated testing is significantly faster than manual testing, especially for larger numbers of test cases. Additionally, the gap between the two lines widens as the number of test cases grows, emphasizing the efficiency gains of automation.

However, the above chart only showcases the ideal setting where the experiment expects software testers to perform 100 test cases in one go. Below table showcases the estimated realistic time that manual tests would take by introducing rest time and off-work hours. Rest time refers to 5 minutes (300 seconds) break for software testers after every 10 test cases, leading to 3000 seconds of pause for every 100 test cases. Off-work hours refer to the 16 hours (57600 seconds) rest time after the conventional 8 hours day job for an employee. Thus, when the time taken for manual testing exceeded 8 hours, which is 28,800 seconds, it would account for an additional 57,600 seconds of rest time. The total manual time for each thousands of test sets is calculated by taking both rest time and off-work hours into account, as shown in Table 23.

Number of test cases	**Automation**	**Manual**	**Manual rest time (5 minutes per 10 test cases)**	**Manual off work hours (16 hours after 8 hours of day job)**	**Total manual time**
100	3470	5525	3000	-	8525
200	6940	11050	6000	-	17050
300	10410	16575	9000	-	25575
400	13880	22100	12000	-	34100
500	17350	27625	15000	-	42625
600	20820	33150	18000	57600	108750
…	…	…	…	…	…
1,000	34700	55250	30000	57600	142850

Table 23: Estimated duration as the number of test cases increases with resting time.

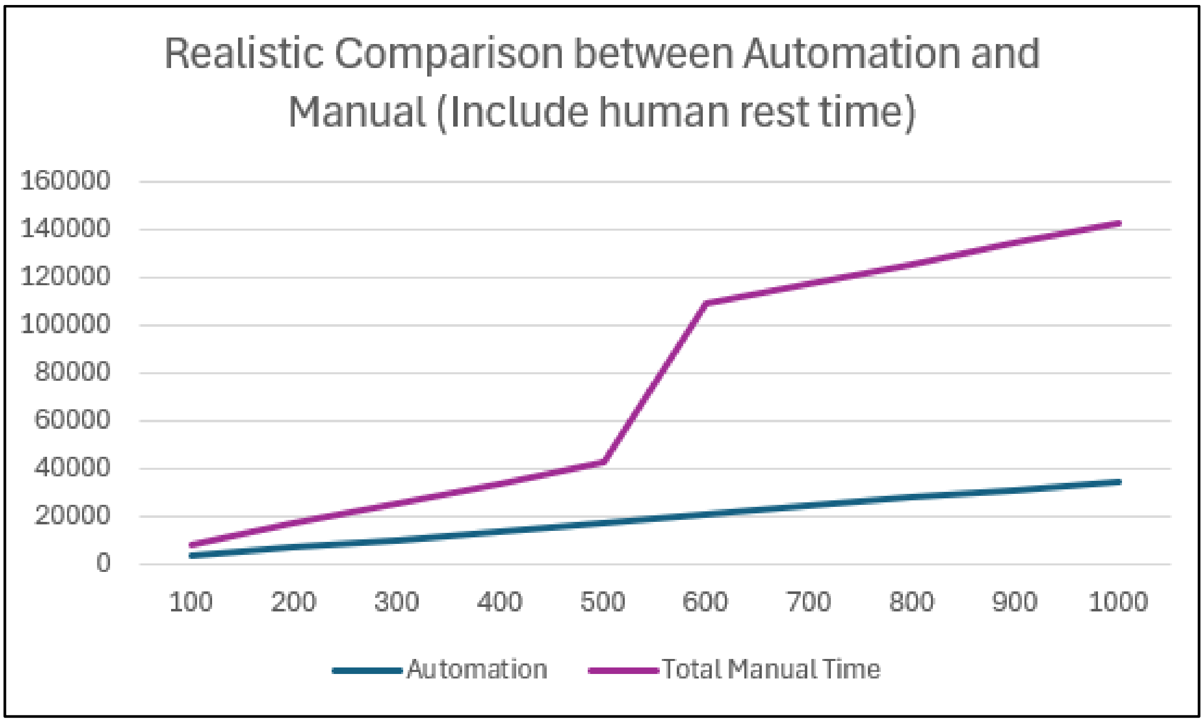


Figure 31: Realistic comparison between automation and manual testing

Based on Figure 31, the spike observed between 500 and 700 test cases is due to the incorporation of human off-work hours for manual testing. At 500 test cases, manual testing is still within the 8-hour workday limit. Beyond 500 test cases, the cumulative time required begins to exceed 28,800 seconds. Therefore, between 500 and 700 test cases, the off-work hours rest period (57,600 seconds) is introduced, causing a significant spike in the total manual testing time.

As a result, manual testing significantly takes more time to complete all test cases as compared to automation testing due to factors such as resting time and off-work hours, portraying the higher productivity of using automation instead of human labor ("A Case Study of Test Automation in Agile Software Development", 2023).

4.2.4 Scalability

Based on a conversation with QueueBee, it is mentioned that most of the time there are multiple deployments within the same timeframe for different clients. As a result, the employees had to test multiple softwares which might be stressful and time-consuming. In this section, a simulation is conducted by using the data of 6000 test cases from 4.2.4 Productivity, as shown in Table 24.

Number of test cases	**Automation**	**Manual**	**Manual rest time (5 minutes per 100 test cases)**	**Manual off work hours (16 hours after 8 hours of day job)**	**Total manual time**
…	…	…	…	-	…
600	20820	33150	18000	57600	108750
…	…	…	…	…	…

Table 24: Total time for test cases.

- **Experiment results**

The simulation is carried out by calculating the total time needed to perform 10 of these 600 test cases to accommodate for 10 different clients, as shown in Table 25.

Number of applications	**Number of test cases (600 test cases per application)**	**Automation (seconds)**	**Manual (seconds)**
1	600	20,820	108,750
2	1200	20,820	217,500
3	1800	20,820	326,250
4	2400	20,820	435,000
5	3000	20,820	543,750
6	3600	20,820	652,500
7	4200	20,820	761,250
8	4800	20,820	870,000
9	5400	20,820	978,750
10	6000	20,820	1,087,500

Table 25: Estimated test case for number of application

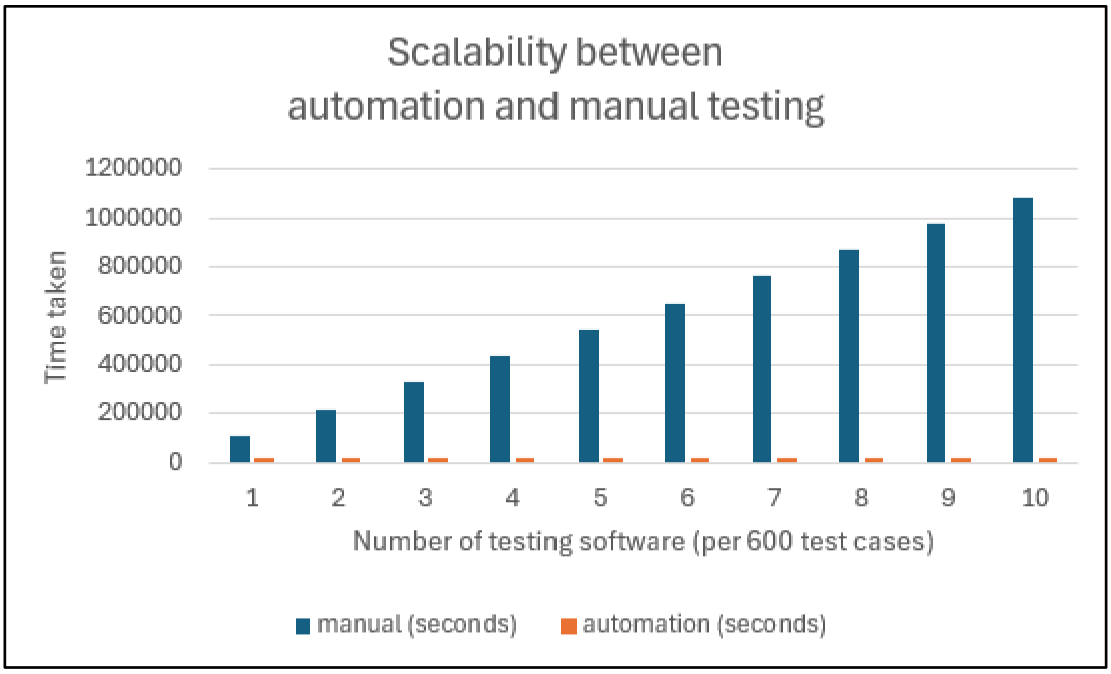


Figure 32: Scalability between automation and manual testing

Based on Figure 32, it is evident that despite the number of software applications being tested, the total duration for automation testing remains constant at 20,820 seconds (5.8 hours). Meanwhile, the duration for manual testing exhibits a linear growth, increasing from 108,750 seconds (30.2 hours) for a single software application to 1,087,500 seconds (302 hours) for ten software applications.

In other words, when comparing the testing time for a single software with 600 test cases, manual testing takes roughly five times longer than automation testing. When comparing the testing time for ten software applications with a total of 6,000 test cases, manual testing takes 50 times longer than automation testing. This contrast highlights the significant scalability advantages of automation testing over manual testing (Sheikh, 2022).

- **Discussions**

The provided chart effectively illustrates the huge difference in scalability between manual and automation testing when dealing with multiple software deployments within a short timeframe. This scenario is particularly relevant to QueueBee, given their need to conduct testing for various clients simultaneously.

As the number of software increases, the manual testing time grows linearly. This is due to the sequential nature of manual testing, where each software requires dedicated time and resources. In contrast, automation testing time remains relatively constant regardless of the number of software. This is because automation enables parallel execution of test cases, significantly improving efficiency. The chart clearly demonstrates the substantial time savings achieved through automation. As the number of software applications increases, the gap between manual and automation testing time widens exponentially, which is why automation is much more scalable than manual testing.

4.2.5 Consistency

To evaluate the consistency between automated and manual testing methods, the experiment was conducted three times under different external environments, which are quiet, noisy, and multi-tasking. The goal is to observe whether the time taken for each test case would remain the same under different settings.

- **Experiment results**

Table 26 summarizes the time taken for both manual and automated testing across different environments.

Test case	**Quiet**		**Noisy**		**Multi-tasking**	
	Manual (seconds)	**Auto (seconds)**	**Manual (seconds)**	**Auto (seconds)**	**Manual (seconds)**	**Auto (seconds)**
1	9	4	10	4	9	4
2	39	65	38	65	65	65
3	14	63	20	63	45	63
4	8	56	12	56	39	56
5	24	55	30	55	86	55
6	56	65	66	65	98	65
7	65	35	78	35	135	35
8	35	59	59	59	89	59
9	46	72	54	72	96	72
10	34	68	66	68	77	68
11	33	87	54	87	123	87
12	48	34	76	34	67	34
Total duration (seconds)	**416**	**663**	**563**	**663**	**929**	**663**
Average duration (seconds)	**34.667**	**55.25**	**46.9167**	**55.25**	**77.4167**	**55.25**

Table 26: Result of duration with the environment factors.

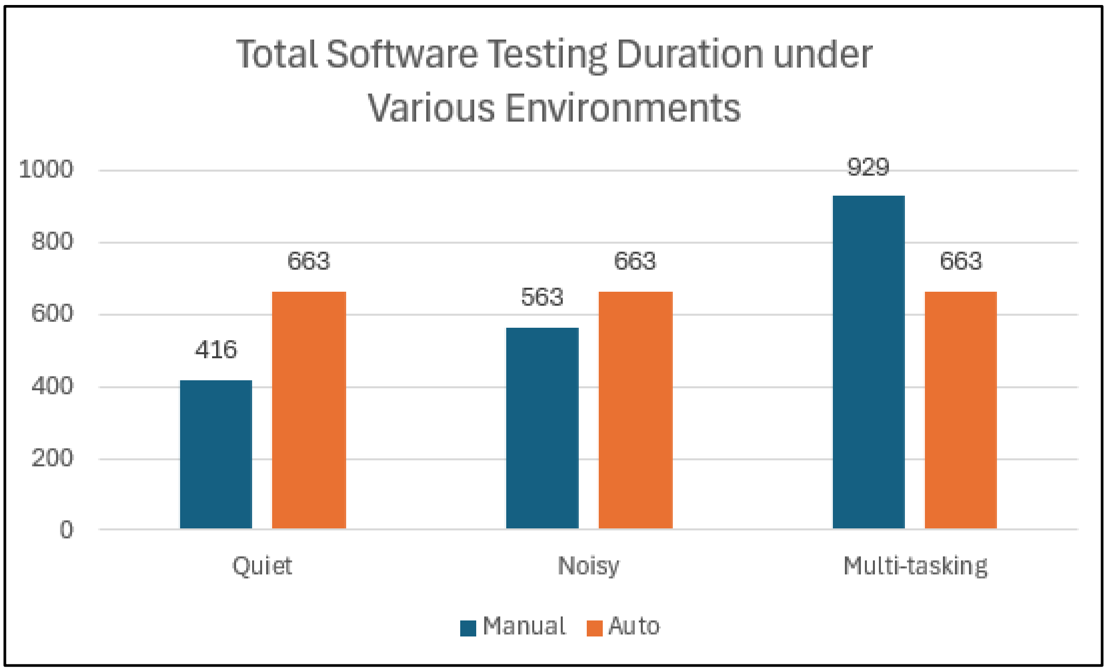


Figure 33: Total Software testing duration under various environments.

As shown in Figure 33, in a quiet environment, manual testing took 416 seconds while automation testing took 663 seconds, whereas in a noisy environment, manual testing took 563 seconds while automation testing took 663 seconds, where the difference is smaller than quiet setting. Meanwhile, in a multi-tasking environment, manual testing took 929 seconds while automation remained at 663 seconds, showing that manual testing took 40% more time than automation under a multi-tasking setting.

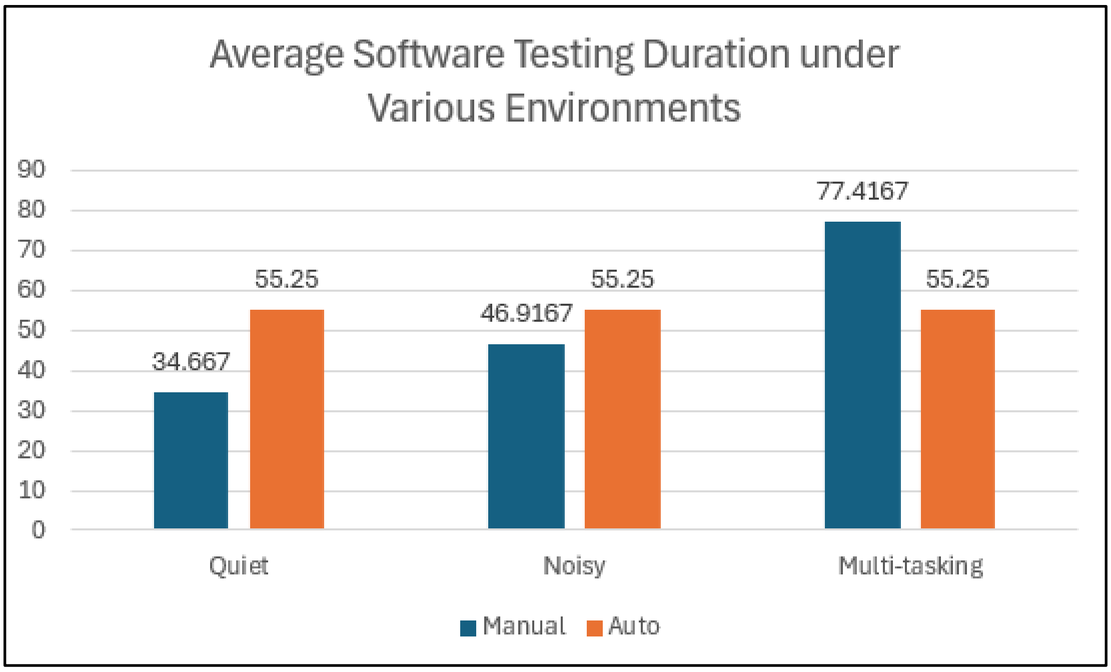


Figure 34: Average Software testing duration under various environments

As seen from Figure 34, in terms of the average time taken, it is obvious that under a quiet environment, the average duration for each test case under manual testing was 34.67 seconds, while automation testing took 55.25 seconds. Meanwhile, in a noisy environment, the average duration for manual testing was 46.92 seconds, with just a 9 second difference from the automation testing at 55.25 seconds. In contrast, under a multi-tasking environment, the average duration for manual testing was 77.42 seconds, which was 22 seconds more than the automation testing which remained at 55.25 seconds.

- **Discussions**

The duration for automated testing remains stable at 663 seconds across all environments. This consistency suggests that automated testing using Selenium is resilient to external environmental factors such as noise and multitasking (Barra et. al., 2022). Since automated tests are scripted and executed by machines, they are less affected by variability caused by human factors, leading to a stable performance regardless of external conditions.

The duration for manual testing increases significantly from a quiet environment (416 seconds) to a noisy environment (563 seconds), and further to a multi-tasking environment (929 seconds). This variability indicates that manual testing is highly susceptible to environmental distractions. When testers are exposed to noise or are required to multitask, their efficiency decreases, leading to longer test durations.

In a quiet environment, manual testing is faster than automated testing, suggesting that manual testing can be efficient when distractions are minimized. However, as environmental distractions increase, the efficiency of manual testing decreases dramatically, whereas automated testing maintains its performance. In the multi-tasking environment, the total duration for manual testing is significantly higher compared to automated testing, highlighting the clear advantage of automation in such conditions.

4.2.7 Discussion summary

As a result, all key metrics suggest that the automation testing system performed much better than manual testing, with significant differences in terms of execution speed, testing accuracy, productivity, scalability, and consistency. The result achieves the second objective of evaluating the performance between automation and manual testing.

4.3 Descriptive analysis of readiness

This section provides a descriptive analysis of the survey about the BPC from the organization in the form of factors. The survey is divided into five factors which are: top management support, internal training, documentation, policy and financial support. The survey implemented a 5-point Likert scale to rate their level of agreement and disagreement in respective to the factor. Table 27 below outlines the data for each key metrics.

Component Descriptive		Mean	S.E	Std Dev	Var
Top Management Support					
1	Top management actively supports the implementation of new technology initiatives.	4.53	0.19	0.74	0.55
2	Top management understands the important of automation technology in software system	4.26	0.23	0.88	0.78
3	The leadership team provides the necessary resources for the success of software automation projects.	4.13	0.29	1.13	1.27
4	There is a clear commitment from top management to drive technological changes in the organization.	4.53	0.16	0.64	0.41
5	Management frequently communicates the importance of adopting new technologies.	4.00	0.17	0.65	0.43
Internal Training					
1	Our organization has a structured internal training program for new technologies.	3.66	0.30	1.18	1.38
2	Employees receive adequate training before implementing new software systems.	3.73	0.32	1.22	1.49
3	Internal training sessions are comprehensive and cater to different learning styles.	3.46	0.31	1.18	1.41

4	Employees feel that the training provided is relevant and up-to-date.	4.06	0.25	0.96	0.92
5	There is a system in place to assess and improve the effectiveness of internal training programs.	3.67	0.37	1.44	2.10
Documentation					
1	The company provides clear and detailed documentation for new software systems.	4.13	0.24	0.92	0.83
2	Employees have easy access to documentation for troubleshooting and learning.	4.06	0.23	0.88	0.78
3	Documentation is regularly updated to reflect changes and new developments.	3.73	0.25	0.96	0.92
4	The documentation provided is easy to understand and follow.	4.07	0.23	0.88	0.78
5	There is a feedback mechanism for employees to suggest improvements in documentation.	4.20	0.30	1.14	1.31
Policy					
1	The company has clear policies that support the adoption of new technologies.	4.26	0.25	0.96	0.92
2	There are procedures to ensure compliance with technology-related policies.	3.71	0.28	1.07	1.14
3	Policies are reviewed and updated regularly to align with technological advancements.	3.86	0.23	0.92	0.84
4	There are rules and regulations covering the whole development of automation system	3.93	0.3	1.16	1.35
5	Employees are aware of the policies regarding the use of software automation	3.93	0.28	1.07	1.15

Financial Support					
1	The company allocates sufficient funds for technology adoption and implementation.	4.33	0.21	0.81	0.67
2	Budget constraints do not hinder the adoption of new technologies.	3.27	0.40	1.53	2.35
3	The financial benefits of adopting new technologies are well-communicated within the organization.	4..13	0.22	0.83	0.70
4	There is a clear financial plan to support the implementation of the QueueBee system.	3.80	0.35	1.34	1.89
5	The organization is willing to invest in necessary tools and resources for technology projects.	4.40	0.19	0.74	0.54

Table 27: Descriptive Analysis of the Survey Question.

4.3.1 Top Management Support

The top management support section received mean scores ranging from 4.00 to 4.53 across the five questions in this section. With an overall mean value of 4.29, the results highlight that respondents agree that top management supports the implementation of automation software, as the mean value is above 4. The respondents felt that top management was keen on technological change and was consistently emphasizing the importance and benefits of the automation software. In addition, it can be interpreted that the respondents felt that top management was decisive in driving this change, as the department heads were provided with the sufficient resources to ensure the successful implementation of the software.

4.3.2 Internal Training

The internal training section received a mean score from 3.46 to 4.06 throughout the 5 questions in the section. With a mean value of 3.72, indicating a descent agreement in internal training provided is adequate as the mean score was below 4. Despite the strong support from top management for the introduction of the software automation system, respondents felt that more emphasis could be placed on internal training. This ensures that employees receiving sufficient and comprehensive training is crucial for the effective implementation of automation and minimizing potential challenges during adoption.

4.3.3 Documentation

The documentation section received mean scores ranging from 3.73 to 4.20 across the five questions. With an overall mean value of 4.04, the results suggest that employees agree that the documentation provided to the team was sufficient. The average score of 4 indicates that respondents found the documentation to be clear, easy to understand, easily accessible and was responsive towards continuous feedback for improvements. However, the respondents only slightly agree that the documentation is not updated as frequently as expected, indicating room for improvement in keeping materials current.

4.3.4 Policy

The policy section received mean scores ranging from 3.71 to 4.26 throughout the 5 questions in the section. With an overall mean value of 3.94, indicating that employees agree that the policies in place support the adoption of new technologies. The average mean value being under 4 suggests that while policies are regularly reviewed and employees have only a slight awareness of the rules and policies related to software automation. Although respondents agree that company policies do generally support the adoption of new technologies, the respondents felt that procedures ensuring compliance with technology-related policies are insufficient, as it reflects a lower score of 3.71.

4.3.5 Financial Support

The financial support section received mean scores ranging from 3.27 to 4.40 across the five questions in this section. With an overall mean value of 3.99, the results indicate that employees agree that the company provides financial backing for the adoption of new technologies. Respondents felt that the company understands the benefits of new technologies and was willing to allocate sufficient funds towards investing in tools to improve efficiency and productivity. However, respondents slightly agree that the company has a clear financial strategy towards the support of new technologies and remain neutral regarding whether budget constraints would impact technology investments.

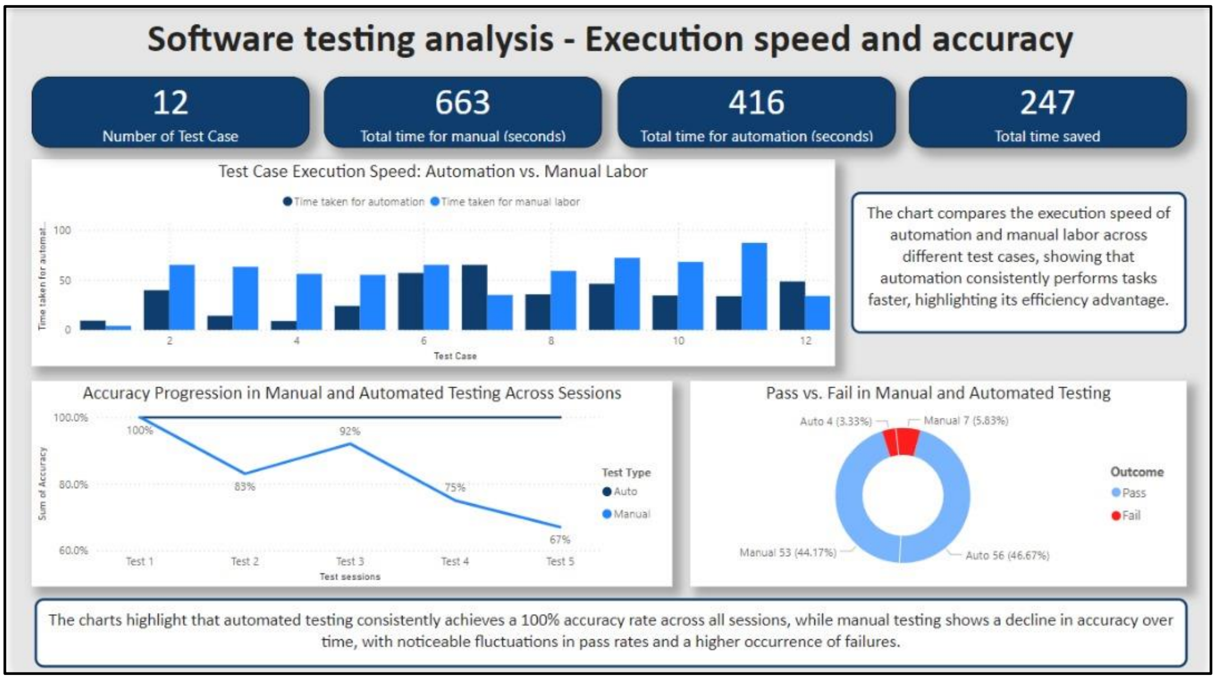
4.3.6 Discussion Summary

As a result, the overall mean value across all five aspects of the questionnaire was 4. This indicates that the respondents “agree” that the company is prepared to introduce the software automation system. The company’s readiness was reflected throughout all five aspects, showing consistent alignment in areas such as: top management support, internal training, policy, financial support, and documentation. The result satisfies the requirement of achieving the third objective by evaluating the company’s readiness to implement an automated software testing system for business use-cases.

4.4 Discussion for business analytics

This section discusses the three main dashboards created using PowerBi as a business analytic tool to visual data and portray useful insights for stakeholders.

4.4.1 Software testing analysis for execution speed and accuracy



The chart compares the execution speed of automation and manual labor across different test cases, showing that automation consistently performs tasks faster, highlighting its efficiency advantage.

The charts highlight that automated testing consistently achieves a 100% accuracy rate across all sessions, while manual testing shows a decline in accuracy over time, with noticeable fluctuations in pass rates and a higher occurrence of failures.

Figure 35: PowerBi dashboard for execution speed and accuracy

The above Figure 35 offers a visual comparison of productivity metrics between automated and manual testing, focusing on total testing time, ideal and realistic productivity scenarios, and the time saved with automation. Here's how the dashboard can enhance business analytics for stakeholders:

- **Key Insights Visualization**:

The dashboard showcases execution speed comparisons between automation and manual testing, along with accuracy trends in a line chart and pass/fail rates in pie chart. The top-level metrics are displayed effectively, giving stakeholders a quick understanding of the performance differences between manual and automated processes.

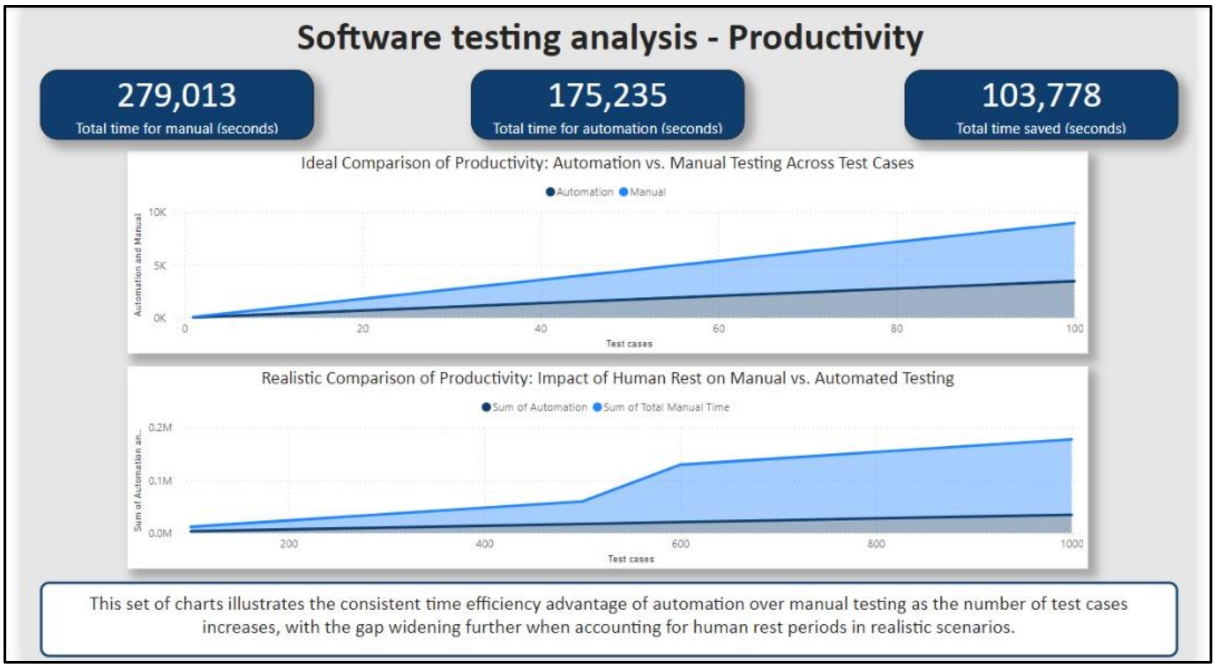
- **Comparative Analysis**:

Visual comparisons of execution speed across test cases and accuracy progression across sessions reveal the advantages of automation's speed and reliability. Additionally, the pass/fail analysis offers a straightforward look at the success rates, highlighting the greater consistency of automated testing.

- **Strategic Decision-Making**:

The data provided in this dashboard equips stakeholders with insights on performance optimization, allowing them to focus on areas where automation can reduce errors and enhance software quality. The detailed comparison of outcomes encourages further investment in automation for long-term gains.

4.4.2 Software testing analysis for productivity



This set of charts illustrates the consistent time efficiency advantage of automation over manual testing as the number of test cases increases, with the gap widening further when accounting for human rest periods in realistic scenarios.

Figure 36: PowerBi dashboard for productivity

- **Key Insights Visualization**:

Figure 36 effectively showcase productivity metrics between manual and automated testing, providing clear and actionable insights. The top-level metrics such as "Total time for manual" vs. "Total time for automation" and the "Total time saved" give stakeholders a quick, high-level view of the overall productivity gain from automation.

- **Comparative Analysis**:

The visual comparison between automation and manual testing highlights the efficiency advantages, which can help stakeholders make data-driven decisions regarding resource allocation and process improvement. The second chart's realistic comparison of productivity considering human rest periods is essential for understanding how automation can reduce downtime and improve throughput.

- **Strategic Decision-Making**:

By presenting data that clearly illustrates time savings and increased productivity, this dashboard empowers stakeholders to justify investments in automation tools and technologies. The realistic projections add credibility to the analysis, making it easier for stakeholders to relate to real-world scenarios and plan for operational improvements.

4.4.3 Software testing analysis for scalability and consistency

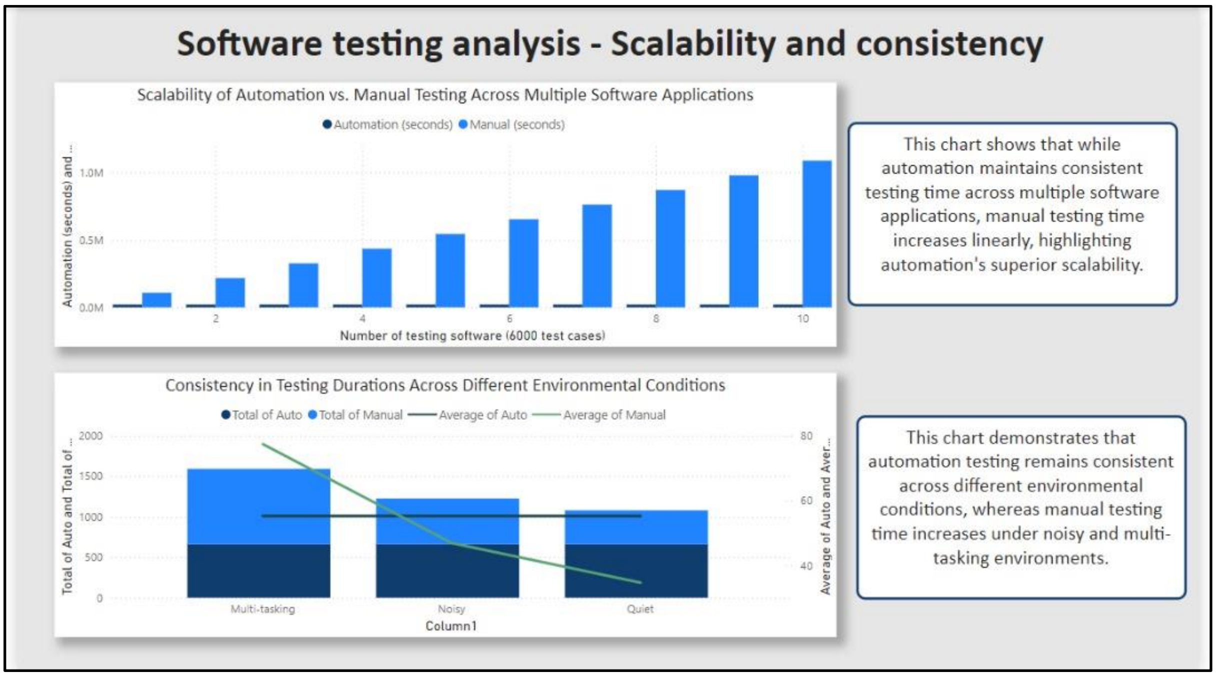


Figure 37: PowerBi dashboard for scalability and consistency

- **Key Insights Visualization**

Figure 37 presents a clear view of scalability in testing, showing how automation maintains consistent time performance across multiple software applications. Additionally, it highlights how testing durations remain stable in different environmental conditions, providing a complete picture of automation's reliability.

- **Comparative Analysis**

The comparative analysis of automation vs. manual testing underlines automation's ability to handle increasing workloads efficiently, while manual testing shows a linear increase in time. The dashboard also contrasts the consistency of automation and manual testing across multitasking, noisy, and quiet environments.

- **Strategic Decision-Making**

Stakeholders can use this data to make informed decisions on implementing automation in large-scale projects and challenging operational environments. By demonstrating automation's scalability and consistency, the dashboard supports strategies for expanding automation across more extensive testing operations without compromising performance.

CHAPTER 5

CONCLUSION

5.0 Chapter Overview

The following chapter illustrates the research summary, research contributions, limitations, and suggestions for future research. All three research objectives were successfully accomplished, providing insights for researchers and business analysts. This research contributes to the current field of software testing, automated testing, business analytics, and Generative AI. It also contributes to other industries and firms as this research showcases benefits of automation, encouraging to automate tasks where beneficial. The limitations of this study are time and resources, hence it was not feasible to include the 'cost' metric. Furthermore, it was not reasonable to develop an end-to-end automated software testing system. It was also not feasible to incorporate Generative AI in each test case due to time and resource limitations. Future studies could explore these components and integrate them into their research for queue management systems.

5.1 Research summary

This summary provides an overview of the objectives and summarizes the findings. The three objectives are discussed below.

5.1.1. Research Objective 1

The first objective was to develop an automated software testing system using Generative AI. This objective was achieved successfully as an automated software system was developed for QueueBee, which provides significant theoretical benefits over manual testing. While Generative AI was utilized, it could have been further improved by incorporating Generative AI in more test cases. However due to the complexity and time-intensive nature of this research, the study was limited to incorporating Generative AI to only one test case. However, a complete automated testing system using Generative AI was developed, hence the first objective has been achieved.

5.1.2. Research Objective 2

The second objective is to evaluate the performance between manual testing and automated software testing systems. This objective was successfully achieved as the performances of the two systems were compared effectively. The evaluation revealed that automated testing performed better than manual testing, with significant differences in terms of execution speed, testing accuracy, productivity, scalability, and consistency.

5.1.3. Research Objective 3

The third objective is to assess the readiness of the company to implement an automated software testing system for business use-case. The readiness of the company was effectively examined through a comprehensive questionnaire, thus successfully achieving the third objective as well. The results revealed that the company is ready to implement an automated software testing system as the respondents agreed on all five aspects. These are top management support, internal training, policy, financial support, and documentation.

5.2 Research contributions

Furthermore, this research contributes to the broader field of software testing by showcasing a practical application of automated testing tools. This research provides a detailed case study on the implementation process, solutions developed, and limitations to provide valuable insights for other companies considering similar transitions in the future. This research can also serve as a reference for future projects, promoting best practices and standardization in automated software testing. This study contributes to the literature of automated testing systems, business analytics, and Generative AI as well.

Lastly, this research underscores the importance of leveraging automation to improve software testing processes. With the automated software testing system, companies in various industries can ensure the reliability and performance of their software application to support their respective business goals. The successful implementation of automated software testing not only benefits QueueBee but also sets a precedent for other companies to follow, driving innovation and efficiency in the field of software development.

5.3 Limitations and Future Research Directions

Several limitations were encountered during the development of this study. One key limitation was the inability to measure the metric ‘cost’ in monetary terms. In theory, the automation should reduce cost, however costs are also incurred while coding the testing automation as experienced programmers would need to be hired. Due to the complexity of quantifying these expenses, it was not feasible to investigate the difference of cost between spending on coding and spending on manual testing. Future research could focus on measuring the costs required with setting up an automated testing system and compare them with costs incurred in manual testing.

Additionally, this research introduced Generative AI in the form of image recognition and validate test cases using prompt engineering. However, this research only included GenAI in one test case as a proof-of-concept, rather than using it as the main objective. Future research could focus on comparing the time taken by different Generative AI models with different types of prompts and measure accuracy. Only Generative AI models could be tested as well for automated testing, being measured using the same key metrics such as productivity and scalability. Due to limitations in time and resources, developing a full scale automated system was not feasible. Future research could explore developing an end-to-end automated testing system, complete with automatic report generation and automatic report delivery to stakeholders.

CHAPTER 6

REFERENCES

Abdellatif, T. M., Capretz, L. F., & Ho, D. (2015). Software Analytics to Software Practice: A Systematic Literature Review. 2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering. https://doi.org/10.1109/bigdse.2015.14

Acito, F., & Khatri, V. (2014). Business analytics: Why now and what next? Business Horizons, 57(5), 565–570. https://doi.org/10.1016/j.bushor.2014.06.001

Akpinar, P., Aktas, M. S., Keles, A. B., Balaman, Y., Guler, Z. O., & Kalipsiz, O. (2020). Web application testing with model based testing method: case study. 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). https://doi.org/10.1109/icecce49384.2020.9179238

Albarka, U. M., & Zhanfang, C. (2019). A Study of Automated Software Testing: Automation Tools and Frameworks. International Journal of Computer Science Engineering (IJCSE). https://doi.org/10.5281/zenodo.3924795

Almansoori, M. T. S., Rahman, I. A., & Memon, A. H. (2021). Correlation between the management factors affecting pmo implementation in uae construction. International Journal of Sustainable Construction Engineering and Technology, 12(3). https://doi.org/10.30880/ijscet.2021.12.03.016

Altmann, S., Ringhof, S., Neumann, R., Wöll, A., & Rumpf, M. C. (2019). Validity and reliability of speed tests used in soccer: a systematic review. Plos One, 14(8), e0220982. https://doi.org/10.1371/journal.pone.0220982

Analytics steps. Analytics Steps. Retrieved June 17, 2022, from https://www.analyticssteps.com/blogs/overview-descriptive-analysis

Aniche, M. (2022). Effective Software Testing: A developer's guide. Simon and Schuster.

Avritzer, A., Britto, R., Trubiani, C., Camilli, M., Janes, A., Russo, B., … & Chalawadi, R. K. (2022). Scalability testing automation using multivariate characterization and detection of software performance antipatterns. Journal of Systems and Software, 193, 111446. https://doi.org/10.1016/j.jss.2022.111446

Bandi, A., Adapa, P. V. S. R., & Kuchi, Y. E. V. P. K. (2023). The power of Generative AI: a review of requirements, models, Input–Output formats, evaluation metrics, and challenges. Future Internet, 15(8), 260. https://doi.org/10.3390/fi15080260

Barra, P. D. l., Luna-Navarro, A., Prieto, A., Vásquez, C. C., & Knaack, U. (2022). Influence of automated façades on occupants. Journal of Facade Design and Engineering, 10(2), 19-38. https://doi.org/10.47982/jfde.2022.powerskin.02

Basak, S. and Hosain, M. S. (2014). Software testing process model from requirement analysis to maintenance. International Journal of Computer Applications, 107(11), 14-22. https://doi.org/10.5120/18795-0147

Ben-Joseph, K. (2021, August 26). *Purposive Sampling 101*. Alchemer. https://www.alchemer.com/resources/blog/purposive-sampling

Ben-Zahia, M. A., & Jaluta, I. (2014). Criteria for selecting software development models. 2014 Global Summit on Computer & Information Technology (GSCIT). https://doi.org/10.1109/gscit.2014.6970099

Bener, A., Misirli, A. T., Caglayan, B., Kocaguneli, E., & Calikli, G. (2015). Lessons Learned from Software Analytics in Practice. In Elsevier eBooks (pp. 453–489). https://doi.org/10.1016/b978-0-12-411519-4.00016-1

Bhanushali, A. (2023). Impact of automation on quality assurance testing: A comparative analysis of manual vs. automated QA processes. International Journal of Advances in Scientific Research and Engineering, 26, 39.

Bidari, A., Jafarnejad, S., & Alaei Faradonbeh, N. (2021). Effect of Queue Management System on Patient Satisfaction in Emergency Department; a Randomized Controlled Trial. Archives of academic emergency medicine, 9(1), e59. https://doi.org/10.22037/aaem.v9i1.1335

Candea, G., Bucur, S., & Zamfir, C. (2010). Automated software testing as a service. Proceedings of the 1st ACM Symposium on Cloud Computing. https://doi.org/10.1145/1807128.1807153

Cauwenberghe, E. V., Jones, R. A., Hinkley, T., Crawford, D., & Okely, A. D. (2012). Patterns of physical activity and sedentary behaviour in preschool children. International Journal of Behavioral Nutrition and Physical Activity, 9(1), 138. https://doi.org/10.1186/1479-5868-9-138

Chandrasekaran*, G., Neethidevan, V., & Murugachandravel, J. (2019). Impact of unit testing in web automation testing. International Journal of Recent Technology and Engineering (IJRTE), 8(3), 1011-1013. https://doi.org/10.35940/ijrte.c4064.098319

Creswell, J. W., & Creswell, J. D. (2018). Research design: Qualitative, quantitative & mixed methods approaches (5th ed.). Sage.

Dakhel, A. M., Nikanjam, A., Majdinasab, V., Khomh, F., & Desmarais, M. C. (2024). Effective test generation using pre-trained Large Language Models and mutation testing. Information and Software Technology, 171, 107468. https://doi.org/10.1016/j.infsof.2024.107468

Davrondzhon, G., & Arne E.H. (2020). Efficiency Metrics and Test Case Design for Test Automation. https://doi.org/10.1109/qrs-c51114.2020.00015

Deming, C., Khair, M. A., Mallipeddi, S. R., & Varghese, A. (2021). Software testing in the era of ai: leveraging machine learning and automation for efficient quality assurance. Asian Journal of Applied Science and Engineering, 10(1), 66-76. https://doi.org/10.18034/ajase.v10i1.88

De Silva, D., & Gunathilake, M. (2023). A case study of test automation in agile software development. *International Journal of Science and Engineering Applications*, 41–49. https://doi.org/10.7753/ijsea1205.1013

Duan, Y., Cao, G., & Edwards, J. S. (2020). Understanding the impact of business analytics on innovation. European Journal of Operational Research, 281(3), 673–686. https://doi.org/10.1016/j.ejor.2018.06.021

Dustin, E., Garrett, T. and Gauf, B. (2009). Implementing automated software testing: How to save time and lower costs while raising quality. Pearson Education.

Dustin, E., Garrett, T., & Gauf, B. (2009). Implementing automated software testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley Professional.

Dustin, E., Rashka, J., & Paul, J. (1999). Automated software testing: Introduction, Management, and Performance. Addison-Wesley Professional.

Espada, A. R., Gallardo, M. d. M. P., Salmerón, A., & Merino, P. (2015). Using model checking to generate test cases for android applications. Electronic Proceedings in Theoretical Computer Science, 180, 7-21. https://doi.org/10.4204/eptcs.180.1

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2023). Generative AI. Business & Information Systems Engineering, 66(1), 111–126. https://doi.org/10.1007/s12599-023-00834-7

Fraser, G. and Arcuri, A. (2014). A large-scale evaluation of automated unit test generation using evosuite. ACM Transactions on Software Engineering and Methodology, 24(2), 1-42. https://doi.org/10.1145/2685612

Garg, A., & Sharma, D. (2023). Generative AI for Software Test Modelling with a focus on ERP Software. International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT). https://doi.org/10.1109/icaiccit60255.2023.10466102

Garousi, V. and Pfahl, D. (2015). When to automate software testing? a decision-support approach based on process simulation. Journal of Software: Evolution and Process, 28(4), 272-285. https://doi.org/10.1002/smr.1758

Guo, X., Okamura, H., & Dohi, T. (2022). Automated software test data generation with generative adversarial networks. IEEE Access, 10, 20690–20700. https://doi.org/10.1109/access.2022.3153347

Guveyi, E., Aktaş, M. S., & Kalipsiz, O. (2020). Human factor on software quality: a systematic literature review. Computational Science and Its Applications – ICCSA 2020, 918-930. https://doi.org/10.1007/978-3-030-58811-3_65

Halani, K. R., Kavita, N., & Saxena, R. (2021). Critical analysis of manual versus automation testing. 2021 International Conference on Computational Performance Evaluation (ComPE). https://doi.org/10.1109/compe53109.2021.9752388

Horalek, J., Urbanik, P., & Sobeslav, V. (2023). Automated Tests Using Selenium Framework. In Nature of Computation and Communication. ICTCC 2022. Lecture Notes of the

Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 473 (pp. 23–40). https://doi.org/10.1007/978-3-031-28790-9_3

Kamal, M., Ali, S., Nasir, A., Samad, A., Basser, S., & Irshad, A. (2022). An automated approach for the prediction of the severity level of bug reports using GPT-2. arXiv (Cornell University), 2022, 1–11. https://doi.org/10.1155/2022/2892401

Kazimov, T., Bayramova, T., & Malikova, N. (2021). Research of intelligent methods of software testing. System Research and Information Technologies, (4), 42-52. https://doi.org/10.20535/srit.2308-8893.2021.4.03

Kenthapadi, K., Lakkaraju, H., & Rajani, N. (2023). Generative AI meets Responsible AI: Practical Challenges and Opportunities. KDD '23: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/3580305.3599557

Khankhoje, R. (2023). Quantifying Success: Measuring ROI in Test Automation. Journal of Technology and Systems, 5(2), 1–14. https://doi.org/10.47941/jts.1512

Kiplagat, J., Kamaku, P. W. & Paul, S. N. (2020). Influence of automated queue management system optimization on performance of National Cement Company Limited. International Academic Journal of Information Systems and Technology, 2(1), 221-244

Kumar, P. S., & Rao, N. S. S. (2012). Automation of Software Testing in Agile Development - An Approach and Challenges with Distributed Database Systems. Global Journal for Research Analysis, 3(7), 102–104. https://doi.org/10.15373/22778160/july2014/35

Kılınç, H. K., & Keçecioğlu, Ö. F. (2024). Generative Artificial intelligence: a historical and future perspective. Academic Platform Journal of Engineering and Smart Systems, 12(2), 47–58. https://doi.org/10.21541/apjess.1398155

Lagrosen, Y., Chebl, R., & Rios Tuesta, M. (2011). Organisational learning and Six Sigma deployment readiness evaluation: a case study. International Journal of Lean Six Sigma, 2(1), 23–40. https://doi.org/10.1108/20401461111119431

Layman, L. and Vetter, R. (2024). Generative artificial intelligence and the future of software testing. Computer, 57(1), 27-32. https://doi.org/10.1109/mc.2023.3306998

Leotta, M., García, B., Ricca, F., & Whitehead, J. (2023). Challenges of End-to-End Testing with Selenium WebDriver and How to Face Them: A Survey. 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). https://doi.org/10.1109/icst57152.2023.00039

Mahapatra, S., & Mishra, S. (2020). Usage of machine learning in software testing. In Learning and analytics in intelligent systems (pp. 39–54). https://doi.org/10.1007/978-3-030-38006-9_3

Majeed, B., Toor, S. K., Majeed, K., & Chaudhary, M. N. A. (2021). Comparative Study of Open Source Automation Testing Tools: Selenium, Katalon Studio & Test Project. 2021 International Conference on Innovative Computing (ICIC). https://doi.org/10.1109/icic53490.2021.9693066

McKinsey & Company. (2024). What is digital transformation? Retrieved July 8, 2024, from https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-digital-transformation

Melegati, J., & Wang, X. (2021). Surfacing Paradigms underneath Research on Human and Social Aspects of Software Engineering. IEEE Xplore. https://doi.org/10.1109/CHASE52884.2021.00013

Meniberu, A. W. (2015). The elements of business process change. Ethiopian Journal of Business and Economics (The), 5(1). https://doi.org/10.4314/ejbe.v5i1.2

Mittal, A. (2016). The influence of waiting time satisfaction on customer loyalty towards multi-stage services in a full-service restaurant: evidence from india. Bulletin of Taras Shevchenko National University of Kyiv Economics, (183), 22-26. https://doi.org/10.17721/1728-2667.2016/183-6/4

Moodley, K. and James, R. (2024). The adoption of ict and robotic automation systems in the pharmaceutical industry. South African Journal of Information Management, 26(1). https://doi.org/10.4102/sajim.v26i1.1744

Mukred, M., Yusof, Z. M., & Alotaibi, F. M. (2019). Ensuring the Productivity of Higher Learning Institutions Through Electronic Records Management System (ERMS). IEEE Access, 7, 97343–97364. https://doi.org/10.1109/access.2019.2927614

Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing. John Wiley & Sons.

Nah, F. F., Zheng, R., Cai, J., Siau, K., & Chen, L. (2023). Generative AI and ChatGPT: Applications, challenges, and AI-human collaboration. Journal of Information Technology Case and Application Research, 25(3), 277–304. https://doi.org/10.1080/15228053.2023.2233814

National Institute of Standards and Technology. (2010). Updated NIST software uses combination testing to catch bugs fast and easy. Retrieved July 2, 2024, from https://www.nist.gov/news-events/news/2010/11/updated-nist-software-uses-combination-testing-catch-bugs-fast-and-easy

Olaleye, T. O., Arogundade, O. T., Misra, S., Abayomi-Alli, A., & Kose, U. (2023). Predictive Analytics and Software Defect Severity: A Systematic Review and Future Directions. Scientific Programming, 2023, 1–18. https://doi.org/10.1155/2023/6221388

Ooi, K., Tan, G. W., Al-Emran, M., Al-Sharafi, M. A., Capatina, A., Chakraborty, A., … & Wong, L. (2023). The potential of generative artificial intelligence across disciplines: perspectives and future directions. Journal of Computer Information Systems, 1-32. https://doi.org/10.1080/08874417.2023.2261010

Park, Y. S., Konge, L., & Artino, A. R. (2020). The Positivism Paradigm of Research. Academic Medicine, 95(5), 690–694. Researchgate. http://dx.doi.org/10.1097/ACM.0000000000003093

Pomberger, G., Bischofberger, W., Kolb, D., Pree, W., & Schlemm, H. (1998). Prototyping-Oriented Software Development - Concepts and Tools. Structured Programming. 12.

Power, D. J., Heavin, C., McDermott, J., & Daly, M. (2018). Defining business analytics: an empirical approach. Journal of Business Analytics, 1(1), 40–53. https://doi.org/10.1080/2573234x.2018.1507605

Pulapaka, S., Godavarthi, S., & Ding, S. (2024). Introduction to Generative AI. In Apress eBooks (pp. 1–29). https://doi.org/10.1007/979-8-8688-0473-1_1

Purposive sampling 101: Alchemer blog. Alchemer. (2021, May 20). Retrieved June 17,

QueueBee Solutions. (n.d.). QueueBee Solutions. Retrieved June 26, 2024, from https://www.queuebee.com.my

Rafi, N. D. M., Moses, N. K. R. K., Petersen, K., & Mantyla, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. 2012 7th International Workshop on Automation of Software Test (AST). https://doi.org/10.1109/iwast.2012.6228988

Rajbhoj, A., Somase, A., Kulkarni, P., & Kulkarni, V. (2024). Accelerating Software Development Using Generative AI: ChatGPT Case Study. ISEC '24: Proceedings of the 17th Innovations in Software Engineering Conference Article No.: 5, Pages 1 - 11. https://doi.org/10.1145/3641399.3641403

Rawat, A. S. (2021, March 31). What is descriptive analysis?- types and advantages:

Reuter, K., MacLennan, A., Le, N., Unger, J. B., Kaiser, E., & Angyan, P. (2019). A software tool aimed at automating the generation, distribution, and assessment of social media messages for health promotion and education research. JMIR Public Health and Surveillance, 5(2), e11263. https://doi.org/10.2196/11263

Ricca, F., Marchetto, A., & Stocco, A. (2021). Ai-based test automation: a grey literature analysis. 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). https://doi.org/10.1109/icstw52544.2021.00051

Rojas, J. M. and Arcuri, A. (2015). Automated unit test generation during software development: a controlled experiment and think-aloud observations. Proceedings of the 2015 International Symposium on Software Testing and Analysis. https://doi.org/10.1145/2771783.2771801

Ruwhiu, D., & Cone, M. (2010). Advancing a pragmatist epistemology in organisational research. Qualitative Research in Organizations and Management: An International Journal, 5(2), 108–126. https://doi.org/10.1108/17465641011068884

Sabev, P. S., & Grigorova, K. (2015). Manual to Automated Testing: An Effort-Based approach for determining the priority of software test automation. ResearchGate. https://www.researchgate.net/publication/287487827_Manual_to_Automated_Testing_An_Effort-Based_Approach_for_Determining_the_Priority_of_Software_Test_Automation

Sajji, A., Rhazali, Y., & Hadi, Y. (2023). A methodology of automatic class diagrams generation from source code using model-driven architecture and machine learning to achieve energy efficiency. E3S Web of Conferences, 412, 01002. https://doi.org/10.1051/e3sconf/202341201002

Sane, P. (2021). A Brief Survey of Current Software Engineering Practices in Continuous Integration and Automated Accessibility Testing. 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). https://doi.org/10.1109/wispnet51692.2021.9419464

Sangwatthanarat, K. (2021). Generating test scripts for a single page web application based on database schema. Proceedings of the 4th International Conference on Applied Research in Engineering, Science and Technology. https://doi.org/10.33422/4th.icarest.2021.08.192

Schwarz, C., Schwarz, A., & Black, W. C. (2014). Examining the impact of multicollinearity in discovering Higher-Order factor models. Communications of the Association for Information Systems, 34. https://doi.org/10.17705/1cais.03462

Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943. https://doi.org/10.1109/access.2017.2685629

Sheikh, W. (2022). Teaching c++ programming using automated unit testing and test-driven development—design and efficacy study. Computer Applications in Engineering Education, 30(3), 821-851. https://doi.org/10.1002/cae.22488

Sneha, K. and Malle, G. M. (2017). Research on software testing techniques and software automation testing tools. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS). https://doi.org/10.1109/icecds.2017.8389562

Sneha, K., & Malle, G. M. (2017). Research on software testing techniques and software automation testing tools. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS). https://doi.org/10.1109/icecds.2017.8389562

Sutapa, F. a. K. P., Kusumawardani, S. S., & Permanasari, A. E. (2020). A review of Automated Testing approach for software regression testing. IOP Conference Series Materials Science and Engineering, 846, 012042. https://doi.org/10.1088/1757-899x/846/1/012042

Swanepoel, D. W., Mngemane, S., Molemong, S., Mkwanazi, H., & Tutshini, S. (2010). Hearing assessment—reliability, accuracy, and efficiency of automated audiometry. Telemedicine and E-Health, 16(5), 557-563. https://doi.org/10.1089/tmj.2009.0143

Tanaka, K., Monden, A., & Yucel, Z. (2019). Prediction of Software Defects Using Automated Machine Learning. 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). https://doi.org/10.1109/snpd.2019.8935839

Tomczak, J. M. (2024, June 25). Generative AI Systems: A systems-based perspective on generative AI. arXiv.org. https://arxiv.org/abs/2407.11001v1

Tuboalabo, N. A., Buinwi, N. U., Okatta, N. C. G., Johnson, N. E., & Buinwi, N. J. A. (2024). Leveraging business analytics for competitive advantage: Predictive models and data-driven decision making. International Journal of Management & Entrepreneurship Research, 6(6), 1997–2014. https://doi.org/10.51594/ijmer.v6i6.1239

Tyagi, A. K., Fernandez, T. F., Mishra, S., & Kumari, S. (2021). Intelligent automation systems at the core of industry 4.0. Advances in Intelligent Systems and Computing, 1-18. https://doi.org/10.1007/978-3-030-71187-0_1

Vila, E., Novakova, G., & Todorova, D. (2017). Automation testing framework for web applications with selenium webdriver. Proceedings of the International Conference on Advances in Image Processing. https://doi.org/10.1145/3133264.3133300

Weisz, J. D., He, J., Muller, M., Hoefer, G., Miles, R., & Geyer, W. (2024). Design principles for Generative AI applications. CHI ’24: Proceedings of the CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3613904.3642466

Wiklund, K., Eldh, S., Sundmark, D., & Lundqvist, K. (2017). Impediments for software test automation: a systematic literature review. Software Testing, Verification and Reliability, 27(8). https://doi.org/10.1002/stvr.1639

Wilder, C. R., & Ozgur, C. O. (2015). Business Analytics Curriculum for Undergraduate Majors. INFORMS Transactions on Education, 15(2), 180–187. https://doi.org/10.1287/ited.2014.0134

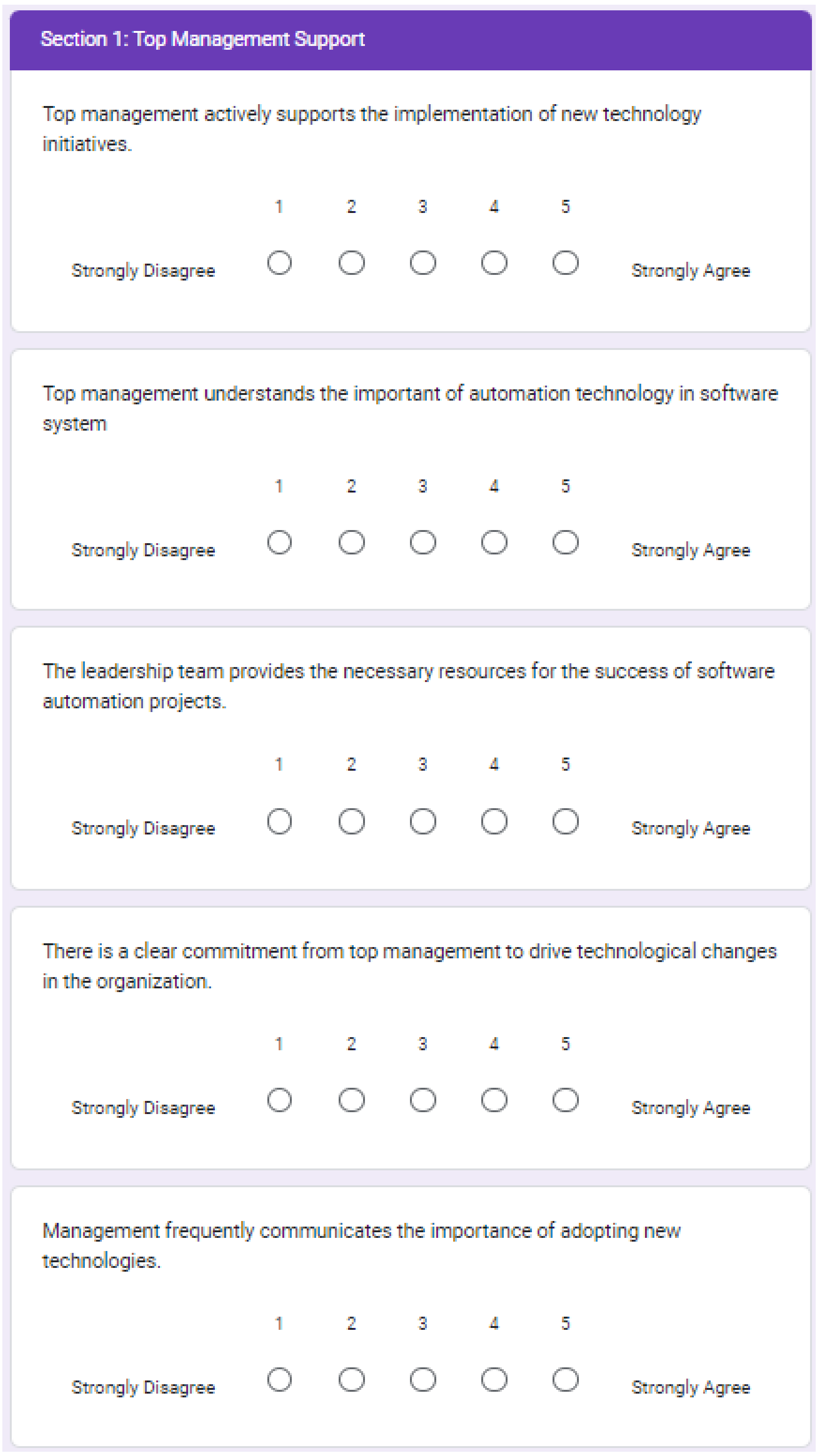
Yadav, V. K., Botchway, R. K., Šenkeřík, R., & Oplatková, Z. K. (2021). Robotic automation of software testing from a machine learning viewpoint. Mendel, 27(2), 68-73. https://doi.org/10.13164/mendel.2021.2.068

Yong, W. K., Maizaitulaidawati M. H, & Suzilawati, K. (2023). Understanding Research Paradigms: A Scientific Guide. Journal of Contemporary Issues in Business and Government, 27(2). https://doi.org/10.47750/cibg.2021.27.02.588

Zhang, K., Song, G., & Kong, J.(2004) Rapid software prototyping using visual language techniques. CiteSeer X (the Pennsylvania State University). https://doi.org/10.1109/iwrsp.2004.1311106

APPENDIX A

Survey Questionnaire



Section 1: Top Management Support

Top management actively supports the implementation of new technology initiatives.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Top management understands the important of automation technology in software system

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The leadership team provides the necessary resources for the success of software automation projects.

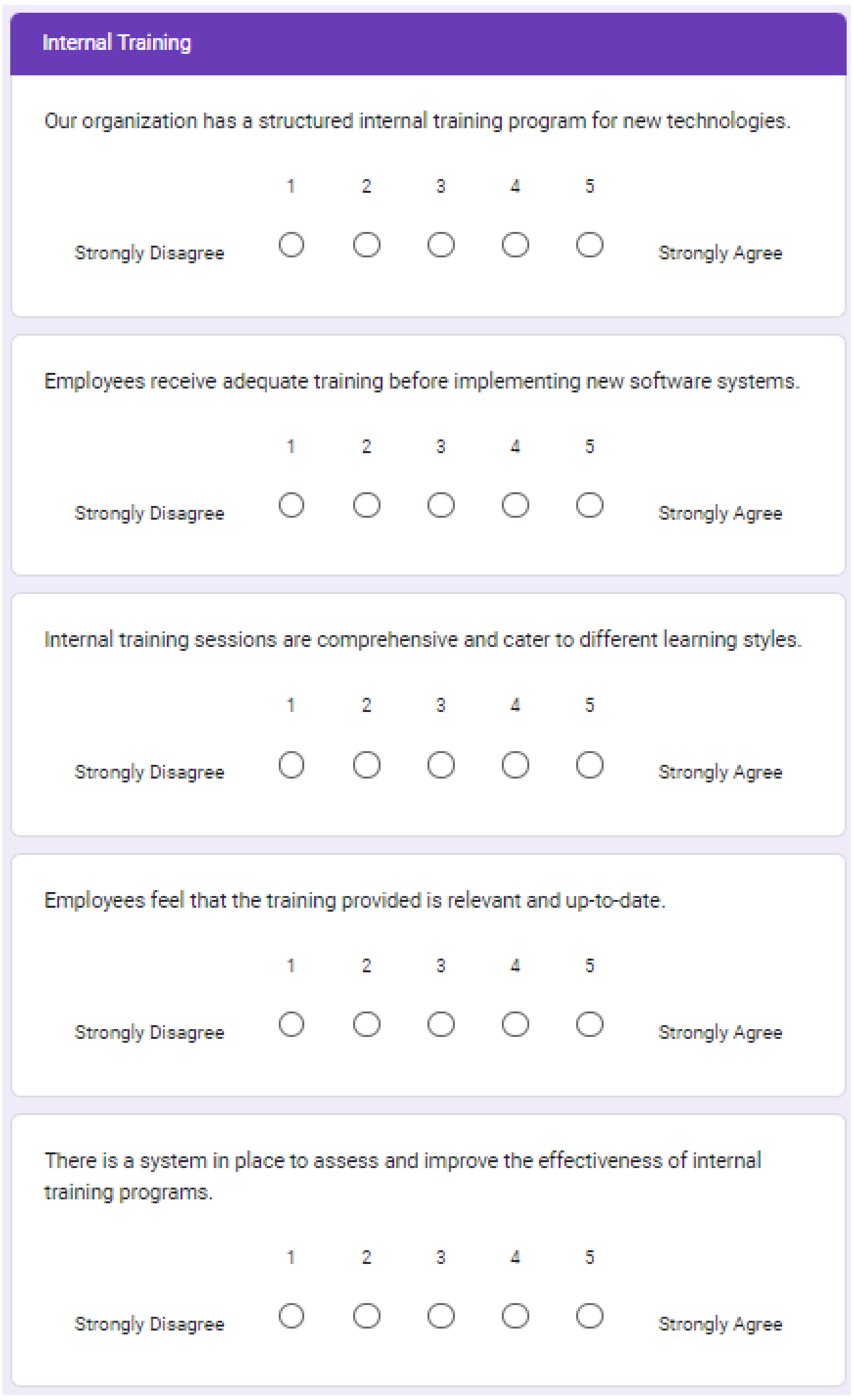
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There is a clear commitment from top management to drive technological changes in the organization.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Management frequently communicates the importance of adopting new technologies.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Internal Training

Our organization has a structured internal training program for new technologies.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Employees receive adequate training before implementing new software systems.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Internal training sessions are comprehensive and cater to different learning styles.

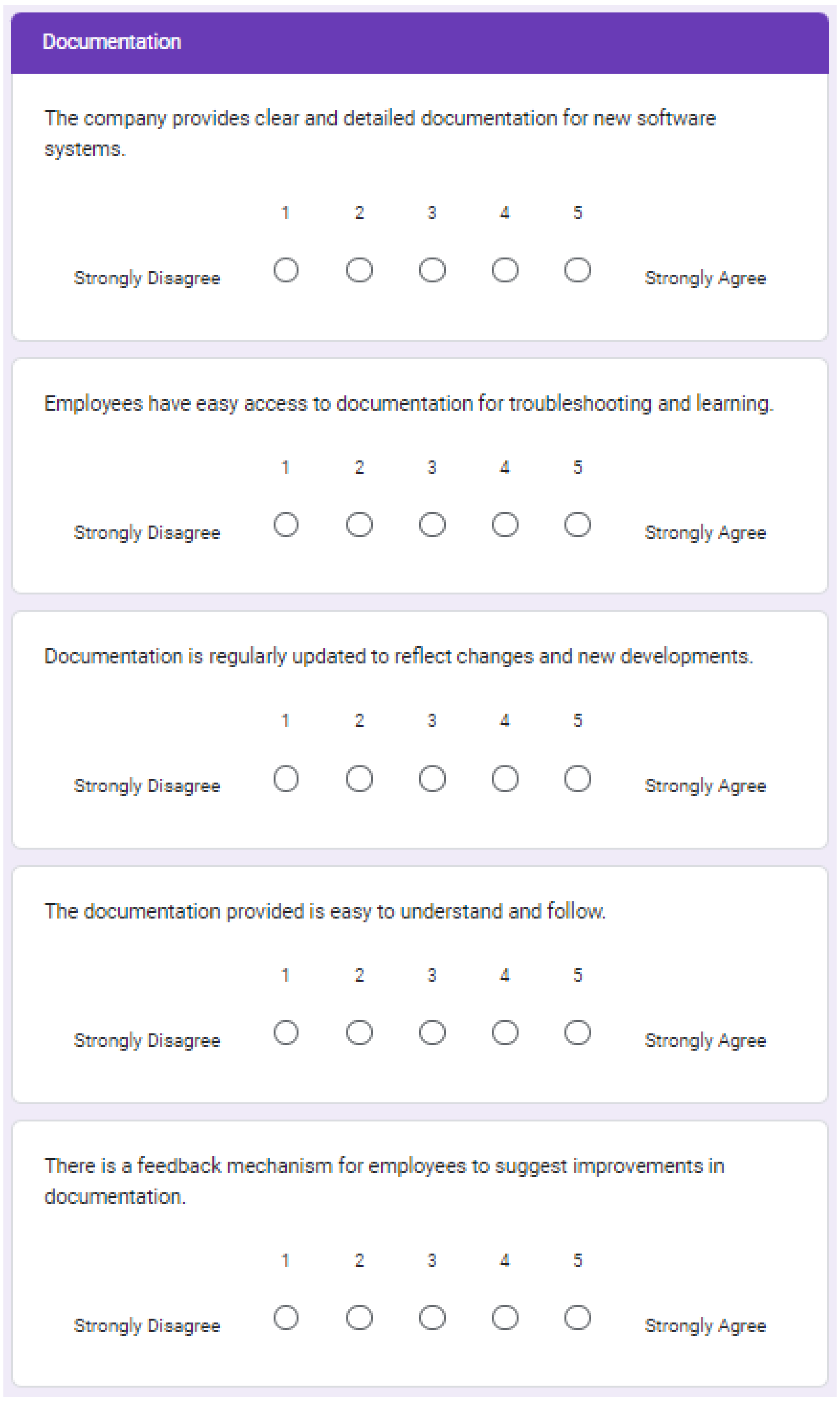
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Employees feel that the training provided is relevant and up-to-date.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There is a system in place to assess and improve the effectiveness of internal training programs.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Documentation

The company provides clear and detailed documentation for new software systems.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Employees have easy access to documentation for troubleshooting and learning.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Documentation is regularly updated to reflect changes and new developments.

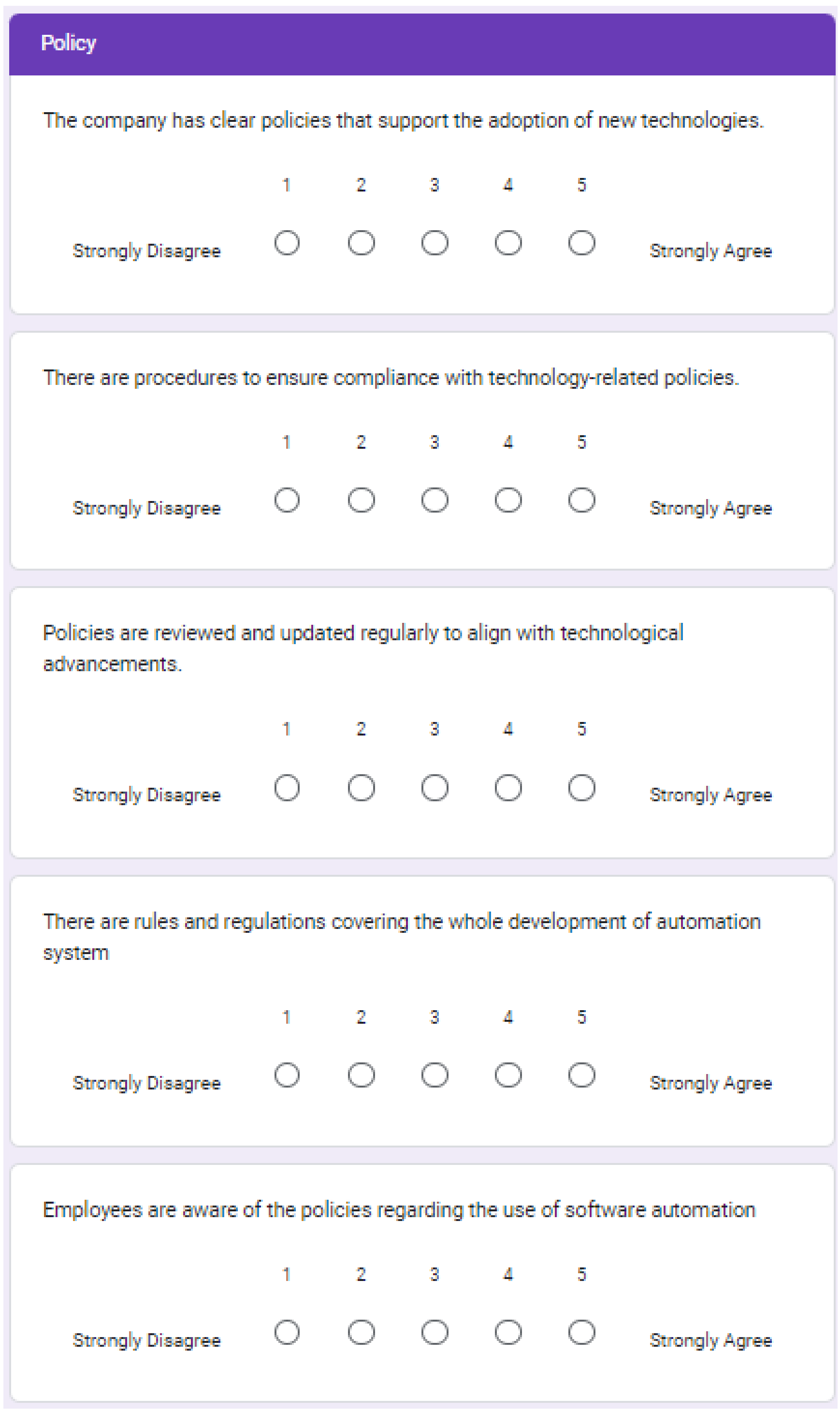
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The documentation provided is easy to understand and follow.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There is a feedback mechanism for employees to suggest improvements in documentation.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Policy

The company has clear policies that support the adoption of new technologies.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There are procedures to ensure compliance with technology-related policies.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Policies are reviewed and updated regularly to align with technological advancements.

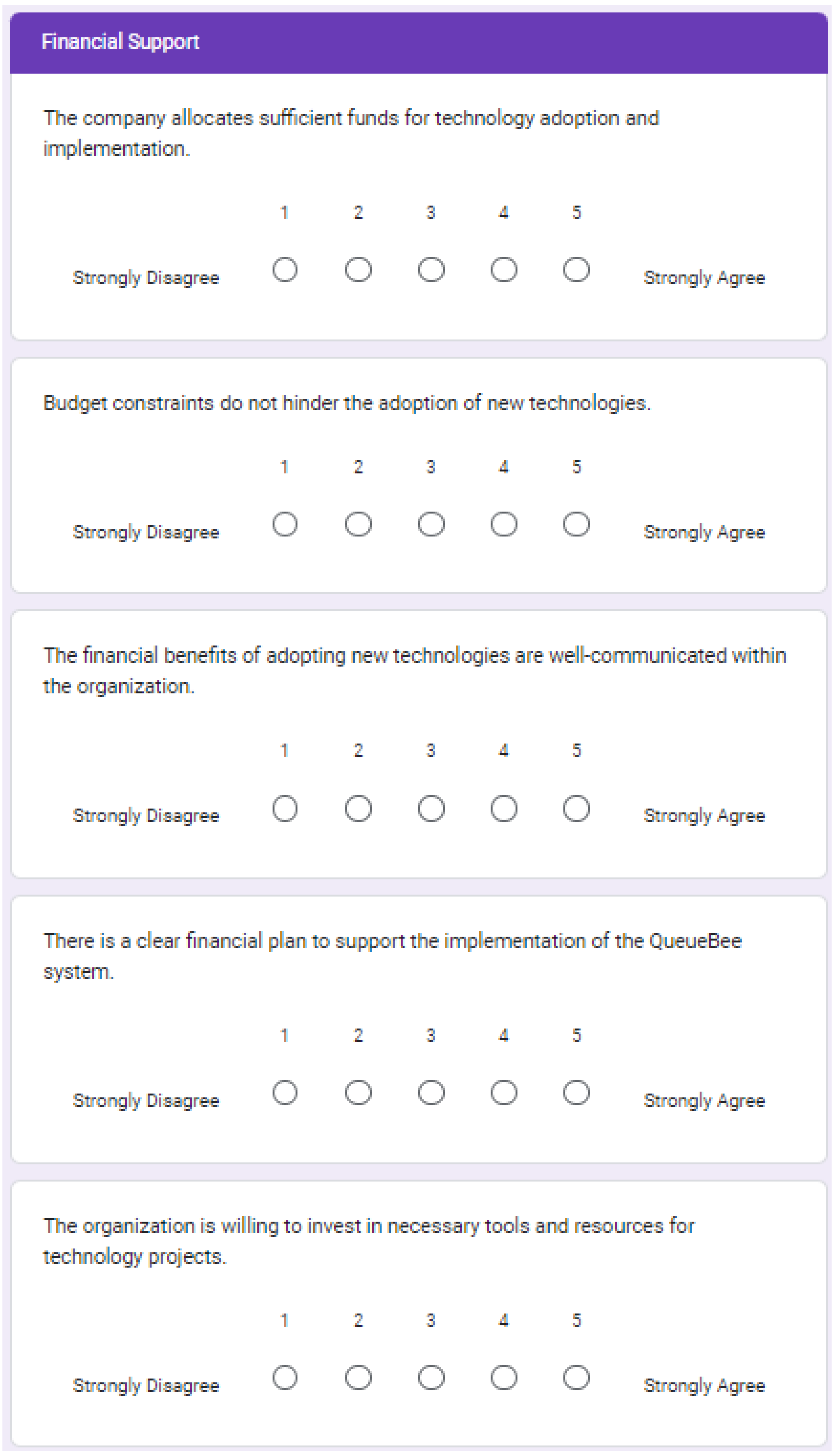
	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There are rules and regulations covering the whole development of automation system

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Employees are aware of the policies regarding the use of software automation

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree



Financial Support

The company allocates sufficient funds for technology adoption and implementation.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Budget constraints do not hinder the adoption of new technologies.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The financial benefits of adopting new technologies are well-communicated within the organization.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

There is a clear financial plan to support the implementation of the QueueBee system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

The organization is willing to invest in necessary tools and resources for technology projects.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree