UNIVERSITY OF GHANA LEGON

DEPARTMENT OF COMPUTER ENGINEERING

SCHOOL OF ENGINEERING SCIENCES

A MULTI-CRITERIA METHOD FOR SELECTING THE RIGHT AUTOMATED SOFTWARE TESTING TOOL

BY

PERPETUAL ADOBEA BIRITWUM

(10934155)

THIS THESIS IS SUBMITTED TO THE UNIVERSITY OF GHANA, LEGON

IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF MPHIL IN COMPUTER ENGINEERING

NOVEMBER, 2024

DECLARATION

I, Perpetual Adobea Biritwum, author of this thesis, hereby certify that this thesis document, except where referenced, is my own work and was carried out under supervision in the Department of Computer Engineering, School of Engineering Sciences, University of Ghana, Legon. This work has never been presented either in whole or in part for any other degree of this university or elsewhere.

…… ……21st November 2024……

Perpetual Adobea Biritwum, 10934155 Date

(Student)

…… ……21st November 2024……

Dr. Nii Longdon Sowah Date

(Supervisor)

…… ……21st November 2024……

Dr. Isaac A. Aboagye Date

(Co-Supervisor)

…… ……21st November 2024……

Dr. Percy Okae Date

(Co-Supervisor)

ABSTRACT

Software testing plays a vital role in the Software Development Life Cycle (SDLC) as it validates and verifies the software being built. To improve efficiency, automation testing has become widely adopted. In the world of software testing, selecting test automation tools can be daunting. Even for professionals, considerable time is often spent identifying and evaluating suitable tools before testing can proceed. However, its success depends heavily on selecting appropriate tools. With countless options available, this selection process often prolongs the testing phase. This study addresses the challenge by identifying key criteria for tool selection and developing a decision-making framework based on the Multi-Criteria Decision Making (MCDM) approach, specifically the Weighted Sum Model (WSM). The framework was implemented as an application that evaluates and ranks potential tools by assigning weights to defined criteria.

The framework was tested on 14 widely used automation tools, including industry-recognized options such as Selenium, Appium, and Cypress, across seven evaluation criteria: ease of use, programming skills, scalability and performance, flexibility and customization, cost and licensing, vendor stability and product support, and reviews and recommendations. To assess effectiveness, feedback was collected from 52 respondents. Findings showed that 67% agreed the framework is useful for the software industry, and 81% considered the selected criteria highly relevant. Additionally, 62% confirmed the framework produced accurate results, while 56% affirmed its efficiency in supporting tool selection decisions. These results demonstrate that embedding an MCDM method in tool selection provides a structured and evidence-based approach. The developed framework not only streamlines decision-making but also contributes to faster and more reliable adoption of automation tools, ultimately improving the overall efficiency of software testing within the SDLC.

DEDICATION

I dedicate this work to my parents, Mr. Edward Dankyi and Mrs. Elizabeth Akrasi, who have encouraged, inspired, supported, and prayed for me throughout my life, education, and career.

ACKNOWLEDGEMENT

I am thankful for the Almighty God's love, protection, and guidance throughout my life and studies. He has supported me every step of the way, and I am truly

My sincere gratitude to Dr. Nii Longdon Sowah and Dr. Isaac Aboagye for their invaluable contributions, advice, and helpful criticism throughout this process. I would like to express my gratitude to every one of the faculty in the Department of Computer Engineering, the Head of Department, Dr. Percy Okae, and the entire graduate committee, for their crucial assistance.

Finally, I would like to express my deepest appreciation to my spouse, Stephen Biritwum, for his unwavering support and encouragement throughout this journey. Many thanks also to my mum, Elizabeth Akrasi, my sister, Deborah A. Dankyi, my brothers, Emmanuel Y. Dankyi and Gideon Omari, my entire family, all my friends, especially Ruth, Suadica, Emma, and Nana K, coworkers, and acquaintances who contributed in various ways. Godspeed to you all.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MCDM	Multi-Criteria Decision Making
MODM	Multi-Objective Decision-Making
MADM	Multi-Attribute Decision-Making
MCDA	Multi-Criteria Decision-Making Analysis
SDLC	Software Development Life Cycle
SUT	Software Under Test
AUT	Application Under Test
App	Application
AHP	Analytic Hierarchy Process
TOPSIS	Technique for Order Preference by Similarity to Ideal Solution
API	Application User Interface
WSM	Weighted Sum Model
IEEE	Institute of Electrical and Electronics Engineers
DSS	Decision Supporting Systems
SSCM	Sustainable Supply Chain Management
OWASP ZAP	Open Web Application Security Project Zed Attack Proxy
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
BWM	Best-Worst Method
ANP	Analytic Network Process
MAUT	Multi-Attribute Utility Theory
SAW	Simple Additive Weighting
SSCM	Sustainable Supply Chain Management
DevOps	Development Operations

CHAPTER 1

INTRODUCTION

1.1 Introduction

As technology advances, the software market advances too. The field of software engineering is changing so quickly that the need for serious, efficient, and effective testing practices has never been greater than now [1]. Software testing has therefore become a necessary and crucial activity in the software industry [2], [3]. In the rapidly evolving landscape of software engineering, the imperative need for robust, efficient, and reliable testing methodologies has never been more pronounced [4]. This is because software can be defective or incomplete until it passes through a reliable testing process [5].

As software systems grow in complexity and scale, the traditional paradigms of manual testing are increasingly proving inadequate to meet the demands of modern software development cycles [6]. About 50% of software projects' development time and effort are put in software testing [7], [8]. Automated software testing, which was once a luxury, is now an indispensable component of modern software development practices, promising increased efficiency, improved test coverage, and enhanced software reliability [9]. This is however dependent on the proper selection of testing tools that correspond to an organisational requirement and objectives [10].

Choosing an appropriate software testing instrument depends on various factors and processes [11], [12]. Each phase is distinct and evolves as the project proceeds [2]. Software testers and developers hence face difficulties when choosing the right testing tool that is suitable for their systems [13]. This thesis addresses a critical challenge at the intersection of software quality assurance and decision science: the selection of appropriate automated testing tools using a multi-criteria decision-making approach.

This research's goal is to develop a systematic, objective framework for evaluating and selecting automated testing tools, with a specific focus on API, web and mobile application testing domains. By leveraging the Weighted Sum Model (WSM), a well-established multi-criteria decision-making method (MCDM), this study aims to provide software engineering professionals with a robust methodology for navigating the multifaceted landscape of automated testing tool selection.

1.2 Background

The evolution of automated software testing is inseparably linked to the broader transformations in software development methodologies and the increasing complexity of software systems. To fully appreciate the context of this research, it is essential to examine the concept of software testing, the methods available, and the persistent challenges in selecting appropriate tools.

1.2.1 Concept of Software Testing

Software testing is a critical activity for ensuring software quality, as it verifies and validates that a system functions as intended and meets specified requirements [2], [14], [15]. IEEE defines it as "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or identifies the difference between expected result and actual result" [18]. Other scholars describe it as the dynamic verification of program behaviour against expected outcomes [17], or more simply, as an evaluation of functionality and compliance with both user and technical requirements [19].

In essence, testing assures that software performs reliably and fulfils its intended purpose. Because of this central role in the Software Development Life Cycle (SDLC), it has become indispensable to software delivery and market readiness [16].

1.2.2 Software Testing Methods

Software testing can be conducted either manually or through automation [20], [21], [22].

Manual testing relies on human execution of test cases without tool support [23], [24]. While it remains applicable in certain contexts, it is often slow, resource-intensive, and prone to error [23], [25]. Manual testing is still relevant today, depending on the type of testing to be done [26].

Automated testing, on the other hand, employs tools and scripts to run tests systematically and repetitively [23], [27]. Automation improves speed, accuracy, coverage, and cost-efficiency while reducing the variability associated with human error [14], [28]. Automated software testing has therefore become increasingly important for handling large-scale and complex systems [23].

1.2.3 The Challenge of Tool Selection

While automation enhances the testing process, it also introduces a practical challenge: selecting the right tool. The software testing landscape is crowded with tools that vary in usability, scalability, cost, customization, vendor support, and compatibility. In the absence of a structured selection framework, organizations often rely on subjective judgment, which can result in wasted resources, delayed projects, or inadequate test coverage.

This challenge highlights the need for systematic, evidence-based methods to guide tool selection. Multi-Criteria Decision-Making (MCDM) approaches provide such a framework by enabling structured evaluation of tools against multiple criteria. This study adopts the Weighted Sum Model (WSM), an MCDM method, as the foundation for a decision-support application aimed at improving the reliability and efficiency of automated testing tool selection.

1.3 Research Problem Statement

Software testing is an indispensable activity in modern software engineering [3]. Yet, one persistent challenge remains: selecting the right automated testing tool. With the vast number of tools available, each differing in functionality, cost, usability, and integration capabilities, teams often struggle to identify the most appropriate option for their project [29]. This struggle is compounded by several factors:

1. The large number of available tools in the market.
2. The diversity of testing requirements across different projects and teams.
3. The need to balance multiple, often conflicting criteria.
4. The lack of a standardised, objective method for comparing and evaluating tools.

As a result, tool selection often consumes significant time and resources, sometimes causing project delays due to tool-related issues [30]. More critically, poor tool selection can lead to low test coverage, inefficient defect detection, integration failures, and increased maintenance costs, all of which undermine software quality and extend project timelines. Preferences are further influenced by factors such as project type, size, and duration. Previous studies have also noted high initial costs in automation setup, tool acquisition, and training as recurring barriers [31].

These challenges highlight the need for a systematic, data-driven approach to tool selection.

This research proposes the use of a Multi-Criteria Decision-Making (MCDM) method, specifically the Weighted Sum Model (WSM), to evaluate tools against defined criteria. By developing and implementing a decision-support application based on this method, the study aims to streamline the tool selection process and improve the overall reliability and efficiency of software testing practices.

1.4 Aim and Objectives of Study

1.4.1 Aim

The main aim of this research is to apply a Multi-Criteria Decision-Making (MCDM) method, specifically the Weighted Sum Model (WSM), to support the selection of automated software testing tools. The study seeks to develop a decision-support application that simplifies and accelerates tool selection based on user responses to defined evaluation criteria, making the process more structured, efficient, and reliable.

1.4.2 Research Objectives

1. To develop a systematic decision-making approach using the Weighted Sum Model (WSM) for automated testing tool selection.
2. To identify and prioritise relevant evaluation criteria (e.g., ease of use, scalability, cost) for selecting automated testing tools.
3. To implement and test a prototype application (Python-based and web-based) that operationalises the framework.
4. To evaluate the effectiveness of the proposed method through structured user feedback, scenario-based testing, and comparative analysis.
5. To ensure the framework is adaptable to different organisational contexts and evolving testing requirements.

1.5 Research Questions

For this study, the following research questions were considered:

1. What are the most critical criteria for evaluating automated testing tools for API, web, and mobile applications, and how can these be prioritised?

2. To what extent does applying a structured MCDM method (WSM) improve tool selection compared to ad-hoc or intuition-based approaches?

3. How can the proposed framework be adapted to different organisational contexts and evolving testing requirements?

4. How does the framework impact testing efficiency, decision consistency, and software quality in practical scenarios?

5. What are the potential limitations of the WSM method in this application, and how can these be mitigated?

1.6 Scope of Study

This study focuses on applying the Weighted Sum Model (WSM) as the primary Multi-Criteria Decision-Making (MCDM) technique for selecting automated testing tools. The model is implemented in both a Python program and a web-based prototype for practical validation.

This research covers:

- Functional, integration, unit, security, and performance testing tools for API, web, and mobile applications.
- Tools compatible with popular programming languages such as React, NodeJS, JavaScript, Python, and Java will be considered.
- Open-source and commercial tools available as of 2024
- Evaluation criteria relevant for small to medium-sized software development teams with a focus on the testing teams

The study excludes:

- Specialised testing tools for embedded systems or Internet of Things (IoT) devices
- Custom-built or in-house developed testing tools
- Code coverage tools and test management tools, as the focus is on tools directly used for testing applications under development.

1.7 Motivation of Study

The main motivation for this study is the need for a more objective and systematic approach to automated testing tool selection that can adapt to evolving technology landscapes. The increasing complexity of software systems and the growing importance of automated testing in ensuring software quality are why having a systematic approach for tool selection cannot be taken for granted. Many scholars have conducted research on the different types of tools, the importance of testing tools, etc. Some have also done a comparative analysis on testing tools; however, not enough has been said about the MCDM methods for tool selection.

Thus, there is a need to conduct research that uses a multi-criteria decision-making method for the selection of the right automated testing tool. The potential for improved testing efficiency and effectiveness through better-aligned tool selection processes is why this study is being considered. Looking at the significant time and resources often spent on selecting and implementing inappropriate testing tools, research work done on the use of multi-criteria decision-making methods for the selection of automated testing tools will improve the selection process for a more efficient software testing experience.

1.8 Significance of Study to Development

This research is necessary because it brings an improvement to the already existing work that has been done in line with the use of MCDM for the selection of the right automated software testing tools. Some of the existing work done was based on a comprehensive taxonomy, which will be simplified in this study. The application that will be developed as a result of the study would be beneficial to software developers, testers, and quality assurance engineers, as it gives a method for deciding on the best automated testing tool based on user preferences.

The key significances of the study are listed below:

1. Improved decision-making: By providing a structured method for tool selection, software testing professionals can make more informed and objective decisions.
2. Cost and time savings: A more effective tool selection process can reduce the likelihood of investing in unsuitable tools, saving both time and resources.
3. Enhanced testing efficiency: Selecting the right tool for specific testing needs can lead to more efficient testing processes and improved software quality.

4. Adaptability: The proposed framework can be adapted to keep pace with evolving testing requirements and new tool capabilities.
5. Knowledge contribution: This study will contribute to the body of knowledge in software testing and multi-criteria decision making in software engineering.

By addressing these themes through the lens of automated testing tool selection, this thesis contributes to the ongoing dialogue on how to effectively leverage automation in software quality assurance.

1.9 Thesis Outline

This work is divided into five (5) chapters as follows:

1. Chapter 1- Introduction: The introductory chapter consists of the overview of the study, the statement of problem, objectives of the study, the motivation behind the research, research questions and relevance of work.
2. Chapter 2 - Literature Review: This presents a comprehensive review of existing research on automated testing tool selection and multi-criteria decision-making methods, with a focus on the Weighted Sum approach. It further points out the limitations of reviewed articles.
3. Chapter 3 - Methodology: This covers the details of the research methodology, including the development of the Weighted Sum model, criteria selection, the tools evaluated, and data the collection methods.
4. Chapter 4 - Design Implementation and Testing: This chapter describes the implementation of the proposed method, including evaluations, results analysis, and comparative assessment.
5. Chapter 5 - Conclusion and Recommendation: This final chapter summarises the key findings, discusses the implications of the research, and suggests areas for future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews existing literature related to automated software testing and the selection of testing tools. While extensive studies exist on automated testing in general, research focusing specifically on the structured selection of appropriate tools remains limited. The review therefore highlights theoretical foundations, conceptual perspectives, and empirical findings relevant to this study, with emphasis on decision-making approaches and evaluation frameworks for tool selection.

2.2 Automated Testing

The journey of automated software testing can be traced back to the 1970s, with the advent of unit testing frameworks [32], [33]. However, it wasn't until the 1990s and early 2000s that automated testing gained widespread adoption, driven by the rise of extreme programming and test-driven development methodologies [34]. Contemporary studies have revealed that test automation has become a very relevant topic, due to the heightened interest in that area. There has been growing importance of automation testing tools with a rise in their adoption from 58.5% in 2015-2016 to 64.4% in 2017-2018 [27]. Today, automated testing is an integral part of modern software development practices, particularly within agile and DevOps frameworks.

The complex nature of today's software systems validates the need for test automation, which is very relevant to software evolution. In previous years, the availability of automated testing tools was limited to a few options on the market [21]. Software testers and engineers were forced to choose from the few options available, whether it suited their goals or not. This implied that most of the testing had to be done manually which was time-consuming and sometimes inefficient [30]. A significant milestone in web application testing automation was the introduction of Selenium in 2004, providing a powerful open-source tool for browser automation [35]. The mobile revolution of the late 2000s and early 2010s introduced new challenges and opportunities, leading to the emergence of tools like Appium to address the specific needs of mobile application testing [36].

In recent times, however, there are a plethora of automated testing tools on the market as software development keeps evolving subsequently affecting the software testing process. Navigating through the vast array of testing tools offered for different types of software tests can be a daunting task [11]. However, it is imperative to make the right selection to ensure a streamlined and productive testing process [30]. Software testers and developers need pragmatic tools to achieve quick and error-free results. There are variant testing tools that have been developed for specific types of software, technology, or specific software criteria [7]. The nature of software determines the type of testing tool to choose. In the absence of such tools, an individual may become exhausted with the constant effort geared towards testing and examining the SUT manually. In summary the current landscape of automated testing is characterised by:

1. A diverse array of tools catering to different testing needs.
2. Integration with CI/CD pipelines, enabling rapid feedback and continuous testing.
3. Incorporation of AI and machine learning capabilities.
4. Cloud-based testing solutions offering scalability and flexibility.
5. A growing emphasis on "shift-left" testing thus integrating testing earlier in the development cycle.

2.2.1 Advantages of Automated Testing

1. Reduced cost of testing: By automating many phases of software testing, organisations can eliminate the high cost of manual testing and reduce expenses by implementing more efficient testing processes [9]. Money spent in hiring manual software testing can be saved just by automating key repetitive software testing processes.
2. Time saving: Automated testing tools can run every day, every time, helping to save time and accelerate the development process [37]. All the time spent in manually validating a software and going through repetitive tests can be well utilised by automating the testing processes.
3. Faster feedback cycles: Automated testing tools enable tests to happen throughout the software development pipeline, providing developers with feedback earlier in the process [38]. Automated tests return results faster to help accelerate production.

4. Greater accuracy: Automated testing tools eliminate error-prone manual testing procedures to vastly improve the accuracy of tests [6]. Since automated tests are done with other software applications, the tendency to make human errors are very minimised.
5. Reduced testing burden: By automating test processes, QA teams can spend more time writing new test cases and building new tools rather than endlessly performing the same manual test [39].
6. Increased test coverage: Teams increase test coverage for products and ensure that more features and functionality can be properly tested by completing testing with an automated test suite at a faster pace [40].

2.2.2 Disadvantages of Automated Testing

1. Sophisticated: The process involved in automated testing is a very complex one [41]. These tests require time, effort/energy as well as appropriate resources. They can also be difficult to put into effect. Sophisticated automated tests are also difficult to maintain, which may cause the quality of the test suite to decline [7]. This can defeat the whole purpose of testing. As such, automated testing requires commitment and intelligence due to its sophisticated nature.
2. Expensive: Another challenge associated with automated testing has to do with the fact that it sometimes requires high startup cost for implementation and execution [42]. Also, the cost of maintaining these tests is relatively high. The cost of training people to effectively manage and handle the software is also huge. Nevertheless, if a desirable result is achieved, the amount of money that was invested initially can be recouped.
3. False positives and negatives: In some cases, tests generate false outcomes which is a major challenge for testers. This is because, in a situation where false positive results show that there is a problem which is actually not true, diagnosis becomes extremely difficult [27]. Consequently, fixing the supposed problem poses a challenge to testers.
4. Not universal: Most people prefer automated testing to manual testing. However, one disadvantage of the automated testing is that certain tasks cannot be automated [21]. In such cases, the tester has no other option than to resort to manual testing.

Even though there are pros and cons to using test automation, the pros outweigh the cons in various ways and that is why test automation is necessary for the success of any complex software that is built.

2.2.3 Comparison of Automated and Manual Testing

Automated testing contrasts with manual testing, which requires teams of quality assurance personnel to test a variety of aspects of a piece of software to ensure that they perform as expected before the software is released [10], [19]. While automated testing allows the re-usability of tools and codes, manual testing does not. The right testing method, coupled with the use of appropriate tools ensures a successful testing outcome. On the whole, both manual and automated tests are beneficial, as the shortcomings of one are addressed by the other.

It is worthy to note that not all tests can be automated but then repetitive tests are better done using testing tools for test automation [25]. Therefore, in certain instances, a combination of both techniques is the best method. Whichever method one chooses, the most important thing is that software errors (bugs) are identified and eliminated, in order to prevent a negative outcome [43]. Even though not all projects require automation for testing [44], it is still a better option because it saves time and cost when applicable [26]. This is because automated testing is more effective and efficient [45]. *Table 1* gives some more details.

2.2.4 Checklist for Test Tool Selection

From the advantages of automated testing listed in previous sections, it is seen that test automation is necessary. In the requirement of test automation lies the non-negotiable process of selecting the test automation tools before performing any automated tests. During any project that requires test automation, there are specific steps involved. These include preparing test cases, selecting test tools, producing test scripts and carrying out the test.

Table 1: Comparison of Automated and Manual Testing

Aspect	**Automated Testing**	**Manual Testing**
Execution Speed	Automated testing is significantly faster as it uses scripts to run tests. Multiple tests can be executed in parallel, reducing the time needed for large-scale testing.	Manual testing is slower, as each test case needs to be executed step by step by a human tester. It is time-consuming, especially for repetitive tasks [46].
Accuracy	Highly accurate when correctly implemented. Automation eliminates human errors during repetitive and complex testing processes.	Prone to human errors, particularly in repetitive tasks or complex scenarios that require attention to detail over extended periods.
Cost Efficiency	High initial setup costs due to tool acquisition and script development, but more cost-effective in the long run, especially for regression and large projects [39].There are open-source free versions all the same.	Lower initial costs, but as the project scales or requires repeated testing (e.g., regression), the costs can increase due to the need for additional manpower and time.
Suitability for Repetitive Tasks	Ideal for tasks that require repeated execution, such as regression testing, load testing, and performance testing. Automation handles these tasks with ease and consistency.	Not efficient for repetitive tasks as manual intervention is required each time, making it tedious and error prone. Better suited for exploratory or ad-hoc testing [25].
Test Coverage	Automated testing can achieve broader test coverage as it can execute more test cases across different environments simultaneously, ensuring that no critical paths are missed.	Manual testing is limited in scope due to time constraints. Test coverage can be compromised as certain edge cases might not be explored due to human fatigue or oversight

From the steps involved in automated testing, there is a focus on the selection of test tools as per the aim of this study. Below is a list of checks often done before the acquisition of any automated testing tool.

1. Consent – leadership approval is needed during the planning stage.
2. Calculation of Profit – the desired profit must be determined prior to the execution of the task/project.

3. Pros and Cons Analysis – a thorough analysis of the advantages and disadvantages of testing need to be done before carrying out the task.
4. Resources – list all the appropriate resources (financial, human, material, etc) that would be needed.
5. Costs – initial cost, maintenance costs, support costs, licensing costs, additional fees, if applicable.

2.2.5 Challenges in Automated Testing Tool Selection

The proliferation of automated testing tools in the market, each with its unique features, strengths, and limitations, has changed the automation tool selection process into a complex decision-making challenge [11], [47]. Below are some of the challenges:

1. Technical Diversity: The wide range of programming languages, frameworks, and platforms used in modern software development necessitates tools that can operate across diverse technical environments [48].
2. Evolving Requirements: As software systems and development methodologies evolve, testing requirements change, requiring tools that are adaptable and forward compatible [49].
3. Cost-Benefit Analysis: Organizations must weigh the costs of tool acquisition, training, and maintenance against the potential benefits in terms of improved efficiency and software quality [50].
4. Skill Set Considerations: The learning curve associated with different tools can be steep, and organizations must consider the existing skill sets of their teams when selecting tools [51].

It is in view of these challenges that the multi-criteria decision-making method for the selection of automated testing tools is being considered for this study.

2.3 Multi-Criteria Decision-Making (MCDM) Methods

Multi-Criteria Decision-Making method is a sub-discipline of operations research that explicitly evaluates multiple conflicting criteria in decision making [52]. These methods are designed to help decision-makers solve complex decision-making problems involving multiple, often conflicting criteria, by providing a structured framework for comparison and selection among alternatives [53].

Key characteristics of MCDM methods include the following:

1. Ability to handle both quantitative and qualitative criteria
2. Consideration of multiple, often conflicting objectives
3. Transparency in the decision-making process
4. Flexibility to incorporate decision-maker preferences

2.4 Survey of Existing Related Works

This section will be divided into two main parts. The first part will be a review of the related work on multi-criteria decision-making methods. The second part of the literature review will be focused on research done on how software testing types and automation testing tools are analysed. This is mainly because, in order to use the multi-criteria methods for automation tool selection, the type of test to be done the tools to be evaluated must be considered.

2.4.1 Multi-Criteria Decision-Making Methods and Its Applications

Mitra Madanchian et al. did a study on Multi-Criteria Decision Making (MCDM) Methods [54]. They emphasised how MCDM methods are recognized as essential tools for decision-making involving multiple criteria. In their study, MCDM is highlighted as a revolutionary method in decision-making, utilising mathematical and statistical tools to structure problems and evaluate alternatives based on qualitative and quantitative criteria. This is why this approach was considered for our study; to use a systematic way of selecting the right automated testing tool. The study mentioned that the evolution of MCDM showcases its growing importance across various fields, necessitating a structured approach to decision-making in our case, the selection of the right testing tool.

Asma J. Abdulwareth et al. also did a study on using MCDM methods for selecting software testing tools [30]. The framework developed facilitated the selection of appropriate testing tools by employing AHP and TOPSIS multi-criteria decision-making techniques. These two MCDM methods were appropriate for their study. However, their methodology cannot be adapted in this study. The weights of the tools (alternatives) and criteria were already assigned using the AHP method in their research. They used pairwise comparison for weights determination, but this approach is best for comparing few alternatives. In our study, the Weighted Sum Model was preferred to allow for weights to be determined by the user.

Dikky Suryadia et al. did a study on using Weighted Sum Model Method for location evaluation [55]. They emphasised the critical role of Decision Support Systems (DSS) in aiding managers

to make informed decisions. They outlined the various factors that affect the selection of sites. They applied the Weighted Sum Model Method to systematically evaluate potential locations. This method provided a structured approach to decision-making in manufacturing considering their context. The study successfully demonstrated the application of the Weighted Sum Model in determining the best location for a manufacturing plan. However, the application of WSM was done manually. The process could have been more efficient with the introduction of a framework to run the WSM method. This is to be addressed in the study.

Salman A. Khan et al. did a study on using weighted sum approach for wind turbine selection [56]. The authors proved from their study how the weighted sum method is effective for multi-criteria decision-making. For their work, each criterion was assigned a weight. In our case, each criterion will be assigned a weight in relation to users' preference for each criterion. The approach ensured that all criteria are considered in a coherent decision-making framework, facilitating the turbine selection. The effectiveness of the model was validated through its application to various turbines for their study. Although their study was not about testing tool selection, they employed the WSM which will be the model used for the selection of automated testing tools for this study.

Dr. P. Rengarajan et al. did a study on the use and benefits of using WSM [57]. They used WSM to evaluate the opinions of farmers on crop insurance schemes using six (6) different criteria. The analysis of farmers' opinions utilised the WSM to quantify the influence of various factors on their decision to insure. The authors used the WSM model because it provided a structured approach to evaluations and analysis. Based on the approach of their study, they were easily able to determine the opinions of the farmers and the various influencing factors. In this same regard, for this study, Weighted Sum Model (WSM) will be used to analyse users' choices for selecting specific automated testing tools.

Chinnasami Sivaj et al. did a study on Future Research Opportunities [58]. They utilised the WSM as a systematic approach for decision-making in agriculture. The method helped in evaluating various alternatives based on multiple criteria, which is essential for making informed decisions. A similar process is to be done in this study, this time round, evaluating automated testing tools. The criteria in the WSM analysis were weighted equally, allowing for a balanced evaluation of the alternatives in the agricultural sector, while also considering the potential for more adaptive weighting strategies in future research. The application of the WSM in multi-criteria optimization, highlighted its effectiveness in generating solutions for complex decision-making scenarios. This is one reason that this method has been chosen for our study.

M. Cinelli et al. did a review of features relevant to Multiple Criteria Decision Analysis (MCDA) [8]. In discussing the MCDA methods, the authors pointed out how weighting criteria is a crucial phase in the MCDA process. The paper discussed the implications of using subjective versus objective weighting, emphasising the need for representative weights that reflect diverse stakeholder preferences. Based on their findings, there is a need to come up with an efficient approach that allows for objective criteria weight assignments when the MCDM method is used. The use of the weighted sum model for our study allows for weights to be assigned based on the user's testing needs hence addressing this need identified in their study.

Ananna Paul et al. did a systematic review of Multi-Criteria Decision-Making Methods in Sustainable Supply Chain Management (SSCM) [59]. Various MCDM methods including AHP and TOPSIS were analysed. From the review it is noticed that the Weighted Sum model was not part of the list of methods analysed. This in a way shows the limited research done on WSM for decision making hence our choice of using WSM for this study. Considering the different methods that were analysed, the Analytical Hierarchy Process (AHP) method came out as one of the best tools for making complex decisions. However, for the purposes of our research, using AHP method restricts the dynamic assignment of weights that happens based on the user's preferences as they respond to the criteria-based questions. The weighted sum model is chosen in our case to allow flexibility of weights assignments.

George Yannis et al. did a study on a review on multi-criteria decision-making methods used in the transport sector [60]. In their study, they reviewed various MCDM methods, including AHP and Simple Additive Weighting (SAW) methods as applied to the transport sector. The paper highlighted the SAW method, which is synonymous with WSM, as one of the simplest MCDM techniques. They evaluated alternatives based on a weighted sum of performance scores across various criteria. Considering the evaluation of MCDM methods that were used for this study, the WSM method is simple to use. In comparing AHP (another method used) to WSM, WSM uses direct scores and weights, whiles AHP involves pairwise comparisons to derive weights, which can be more complex. This is why WSM method was chosen for our research to utilise the simple direct process of direct weight assignment.

2.4.2 Analysis and Evaluation of Software Testing Tools

An empirical study of user interface testing tools was done by Elis Pelivani et al [61]. In their study, they examined three major testing tools: Selenium, Katalon Studio, and Unified Functional Testing (UFT), each with unique features and capabilities. The examination of the

three user interface testing tools was done by comparing the features of the tools. Their criteria included test deployment, application under test, scripting languages, programming skills, ease of installation and use, image-based testing, product support, licence type and cost. The criteria used here can be built upon in our study to build a system for selecting the right testing tool. Even though there was a comparison of the automated testing tools with a set of criteria, there was no application/framework developed to aid in the selection of the right testing tool.

Mubarak Albarka Umar did a research on different categories, levels, techniques, and types of software testing [62]. In his study, he compared 4 main software types: Unit testing, Integration testing, System testing and Acceptance testing. These classifications of testing are key and play a role in the taxonomy of testing tools. However, for this study the taxonomy will be revised to accommodate security testing and performance testing. The paper emphasised the importance of early error detection in software development, as costs escalate with later detection. This contributes to the more reason for why automated testing must be employed and subsequently choosing the right tool is important for test automation.

S. K. Izzat et al. did a study on software testing tools and techniques [63]. They reviewed eight (8) automated testing tools including Selenium, Test Complete, Ranorex and Appium detailing their functionalities and suitability for different testing scenarios. The criteria used for the comparison which can be adopted in this study are licence, approving-platform (Mobile or Web), ease of use and programming-skills. The comparison of these tools was done based on features of each tool but there was no framework developed for easy selection of automated testing tools. The criteria used for the automated testing tools comparison can be used for our study however, their study was limited to eight (8) testing tools.

H.K.D. Prabhashi et al. also have done a study on comparative analysis of functional test automation tools [64]. In their research, they compared five (5) test automation tools with similar characteristics such as platform support and the programming language used. The authors indicated that when selecting test automation tools, it is important to consider ease of use, cost, and supporting platforms [64]. They considered unit testing, integrated testing, system testing and user acceptance testing for their evaluation. Their classification, however, does not include security testing. It has become imperative to perform security testing when releasing new software. Hence the type of test classification will be revised in our study.

Dilara Ateşoğulları et al. did a comparative study on automation testing tools [27]. The authors did a comprehensive comparison of twenty-one (21) automation-testing tools based on fifteen

attributes (criteria). The fifteen (15) attributes included: record and reply, multi programming language support and coding, and system support. The criteria 15 criteria considered will be further classified into a concise list to make the process more efficient. Going through a list of twenty-one (21) tools using 15 different criteria will be more time consuming, especially because this process was done manually using the features offered by each tool. An MCDM-based framework will make the process more efficient.

Mubarak Albarka Umar et al. did a study of automation tools and frameworks [22]. They compared five (5) automation tools based on the features and solutions offered by the providers. The authors emphasised that the choice of automation tools significantly impacts the success of testing projects hence our study on how to select the right testing tool using an MCDM method. Some of the criteria used were test deployment platform, application under test, scripting languages, programming skills, and cost. The set of criteria will be revised for our study. Also, their study compared 5 main tools which is quite limiting if there are different types of tests to be conducted.

Vahid Garousi et al. conducted a grey literature review, synthesising insights from industry experts [47]. They noted that automated tool selection is often based on intuitive understanding rather than formal criteria, leading to challenges in finding suitable tools for specific projects. A comprehensive process for tool evaluation and selection was proposed. This however did not use an MCDM method. The analysis involved iterative refinement of criteria, leading to the identification of key factors influencing tool selection. The study highlighted the need for a more rigorous and systematic approach to tool selection. This informs the approach in this study, considering the various criteria to be used for the tool selection and allowing the stakeholders to determine which criteria is of more importance to their testing needs.

Harsh Bajaj wrote an article on how to choose the right automation tool and framework [11]. The tools he considered were UFT, Selenium, Watir and GEB. The parameters and criteria he considered included ease of adoption, tools usage, reporting capabilities, licence cost, ease of support and learning time. The process of selecting the right of the tool, however, was based on checking the comparison table which was limited to the number of tools compared. Also, the approach used meant that more tools will have to be compared manually using their features before selecting the right testing tools. That whole process of manual assessment is what is to be made efficient with the development of the MCDM-based system.

Roslan Ismail et al. did a study on evaluating and testing software testing tools [65]. It was observed that despite existing studies, there is a lack of empirical guidelines for tool selection, particularly necessitating a framework to aid decision-making. The study aimed to propose a framework tailored to the ICT industry, focusing on measurable factors for evaluating and selecting testing tools. The methodology involved identifying influential factors for testing tools selection from existing literature. Although the study was focused on getting the criteria for tool selection from industry experts, there was no systematic way proposed as a means for seamless test automation tool selection.

From the past research done, a key gap identified is the use of manual decision-making processes to compare and select the right automated software testing tool. Another key gap from the literature reviewed, is the unavailability of an application/tool to make the selection of the right automated testing tool seamless, simple, and fast.

2.4.3 Summary of Literature Review

Table 2 below shows a summary of the literature review done in relation to this study.

Table 2: Summary of Related Work

PAPER	AUTHOR(S)	APPROACH	MAJOR RESEARCH FINDINGS
1. Multi-Criteria Decision Making (MCDM) Methods and Concepts [54]	Hamed Taherdoost et al.	Used key MCDM and MODM methods for analysing decision making processed.	MCDM methods utilised both qualitative and quantitative criteria. MODM is used for criteria without finite number while MADM is finite.
2. Toward A Multi-Criteria Framework For Selecting Software Testing Tools [30]	Asma J. Abdulwareth et al.	Used AHP for weighted importance of criteria and TOPSIS for ranking tools.	MCDM Selection method accurately ranked testing tools even in complex scenarios. Experts' feedback indicated that taxonomy was too complex.
3. Evaluating Location Alternatives for a New Manufacturing Plant Using Weighted Sum Model Method [55]	Dikky Suryadia et al.	Utilised the Weighted Sum Model Method in decision making.	WSM was systematic approach for evaluating alternatives. The model aided in providing informed and effective location selection.
4. Multi-Criteria Wind Turbine Selection using Weighted Sum Approach [56]	Salman A. Khan et al.	Utilised weighted sum method, an MCDM for turbine selection.	Identification of five important decision criteria for turbine selection. The proposed methodology effectively selects the most suitable turbine.
5. Farmers' Views On Crop Insurance And Its Benefits Using Weighted Sum Model (WSM) [57]	Mrs. R. Radhika et al.	Multi-stage random sampling technique was applied. WSM was used to make the decision.	The use of the WSM provided a systematic way of getting and analysing the farmer's opinions on the insurance schemes.

6. Future Research Opportunities Agricultural Sector Using Weighted sum method (WSM) [58]	Chinnasami Sivaji et al.	Weighted sum method utilised for multi-criteria decision-making. Machine learning was used for predictions and data analysis.	Comparison of weighted sum method with other methods showed that analysis with WSM model and machine learning for predictions gave them the expected results for their research.
7. How To Support the Application of Multiple Criteria [8]	Marco Cinelli et al.	Used Tiered rules-based approaches like lexicographic for method selection.	MCDA methods were for discrete alternatives and problem types. Reviews features of MCDA methods for recommendation.
8. Sustainable Supply Chain Management and Multi-Criteria Decision-Making Methods: A Systematic Review [59]	Ananna Paul et al.	Various MCDM methods like ANP, AHP and BWM were used for analysing decision making processes.	MCDM methods utilised in supply chain sustainability. Lack of adequate systematic literature analysis on decision.
9. State-of-the-art review on multi-criteria decision-making in the transport sector [60]	George Yannis et al.	MCDM methods like AHP, TOPSIS, MAUT, and SAW were utilised.	Decision-making in transport requires structured evaluations of options. MCDM techniques address limitations of cost-benefit analysis in transport.
10. An Empirical Study of User Interface Testing Tools [61]	Elis Pelivani et al.	Evaluated the features of valuation some software testing tools.	Katalon Studio is slower due to Groovy language. Selenium is a free, open-source framework for functional web app testing.
11. A Study of Software Testing - Categories, Levels, Techniques, And Types [62]	Mubarak Albarka Umar	Evaluated Testing Categories, Levels, Techniques, and Types.	Software testing approaches, levels, techniques, are mainly dependent on factors per project requirements.
12. Software Testing Techniques And Tools: A Review Summary [63]	S. K. Izzat et al.	Tabulated features after analysing and evaluating testing tools with specific	Selenium is one of the most common testing tools. Automated testing is efficient compared to

		criteria.	manual testing.
13. Comparative Analysis of Functional Test Automation Tools [64]	H.K.D. Prabhashi et al.	Analysed the features of functional test automation tools.	Testing tools can be categorized into test management, unit, integrated and functional testing tools.
14. Automation Testing Tools a Comparative View [27]	Dilara Ateşoğulları et al.	Utilised Record and Playback for UI tests with Test Complete.	Various testing tools are dependent on the test type and platform for testing.
15. A Study of Automated Software Testing: Automation Tools and Frameworks [22]	Mubarak Albarka Umar et al.	Used Linear Automation Framework for UI testing.	Different tools can be used for different jobs as the choice of automation tool largely depends on the nature of the software to be tested
16. Choosing the right test automation tool: A grey literature review of practitioner sources [47]	Päivi Raulamo-Jurvanen et al.	Used systematic mapping process was applied for the literature review.	Criteria including technical, external, and team aspects, aids practitioners in choosing the right test automation tool effectively.
17. Choosing The Right Automation Tool and Framework Is Critical to Project Success [11]	Harsh Bajaj	Used hybrid model combining data-driven and modular frameworks for tools comparison and selection.	test automation frameworks like modular, keyword-driven, and hybrid models play a key role in test tools evaluations.
18. Evaluating And Selecting Software Testing Tools: A case study [65]	Ramona Ramli et al.	Did a literature review to develop initial framework and research hypotheses.	Identified cost, type of testing as factors for evaluating and selecting software testing tools.

2.5 Proposed research work

From the literature reviewed and past work done, a simplified classification of testing tools based on the current software testing and development trends will enhance the selection process. Most of the work previously done focused on comparing existing automated testing tools using the product features of some available testing tools. Some other research also used the MCDM method to select testing tools; however, the framework used the AHP and TOPSIS methods.

In view of the innovation of this study, the weighted sum model will be used as the MCDM method for evaluation and selecting the right automated testing tool. The key reason for this choice is because, in the process of selecting the right automated testing tool, the aim is to give the user or tester the preference to indicate which criteria is of importance to their specific needs. By responding to criteria-based questions, weights will be assigned based on the level of importance to respondents. Subsequently, the right automated tool will be recommended based on the user preferences.

In this research, focus will be on the functional testing tools and performance testing tools which is a broader area for load and stress testing. From the work done by past researchers [22], [64], [66], they compared the tools under the categories of various software testing types. The criteria used in their research will be a foundation of this research.

A Python application will be built to implement the selection of automated testing tools using the weighted sum model - an MCDM method. A web version of the implementation of the model will also be explored. With a web version of the application available, software testers or key decision-makers can easily respond to criteria-based questions and receive recommendations for testing tools specific to their projects without spending long hours on research.

CHAPTER 3

METHODOLOGY

3.1 Introduction

In this chapter, the methodology employed in developing a decision-making system for selecting the most appropriate automated testing tool is presented. The approach for selecting the right tool is using the Weighted Sum Model (WSM), a well-established multi-criteria decision-making method. This has been adapted to address the complex challenge of choosing automated testing tools among various testing tools.

The Weighted Sum Model was chosen because it provides a systematic framework for evaluating alternatives (in this case, automated testing tools) based on multiple criteria. It allows us to:

- Consider various criteria simultaneously, such as vendor stability and cost.
- Assign weights to these factors based on their relative importance to users.
- Calculate a composite score for each tool, enabling objective comparison.

Throughout this chapter, how the WSM is integrated into each phase of the methodology will be discussed. The model's flexibility allows us to translate user preferences (gathered through a series of questions in Appendix B) into quantitative weights, which are then used to calculate scores for each testing tool. This approach helps us to transform what is often a subjective decision into a more objective, data-driven process.

The methodology chapter covers the following:

1. System Design: The overall architecture of the decision-making system, detailing its components and their interactions.
2. Requirements Analysis: This phase involves identifying and specifying the functional and non-functional requirements of the system, ensuring that it meets the needs of its users.
3. Data Gathering: The process of collecting and analysing data on various automated testing tools, as well as defining the criteria for their evaluation is described in this section.

4. Theoretical Framework: Here, the mathematical underpinnings of the WSM and its application in the context of this study is discussed.
5. Development Process: This section outlines the steps taken to implement the system, from initial design to final testing.
6. Modelling and Simulation: This section explains how the decision-making process is modelled and simulations conducted to validate the system.
7. Tools and Materials: Lastly, an overview of the development tools and resources used in creating the system is provided.

In the following sections, each phase of the research methodology is elaborated, providing detailed insights into the approach, its implementation, and how the Weighted Sum Model is leveraged at each stage to arrive at a final recommendation of the right testing tool.

3.2 System Overview

In software testing, automated testing enhances the process of checking the quality of any software built. The difficulty in selecting the right automated testing tool is why this work was done to explore a possible solution to enhance the tool selection process. The key aim of this research is to use an MCDM method specifically, Weighted Sum Model (WSM) to determine the best automated testing tool when there are different alternatives of testing tools.

Automated testing tool selection is one of the most critical and sensible decisions during software development projects [11], which has a considerable effect on effectiveness, quality, and costs. Since there is such a variety of tools in the marketplace, each having its advantages and disadvantages, a systematic approach is required to ensure that the tool which is finally chosen meets the specific needs of the project and software testing requirement.

An overview of the decision-making system developed for selecting the right automated testing tool using the Weighted Sum Model (WSM) multi-criteria method is discussed here in this section. The test automation tool selector (system developed) is designed to address the complex challenge of choosing the right testing tool from the myriad of options available in the market, each with its own strengths and limitations.

The system or application aims to streamline and objectify the selection process by considering multiple criteria simultaneously and weighing them according to the user's responses to

criteria-based questions related to their testing or project needs. The automation tool selector has two main modules: The user interface and the backend that supports the user interface.

3.2.1 Operation of test automation tool selector

Users or test professionals are asked to respond to questions as indications of their preferences for selecting automated selecting a particular testing tool via the user interface. These questions are based on the defined testing tool selection criteria for this study. Based on the user's responses, the backend of the application assigns weights attached to different criteria. These weights are used in the WSM application to determine the best tools for recommendation.

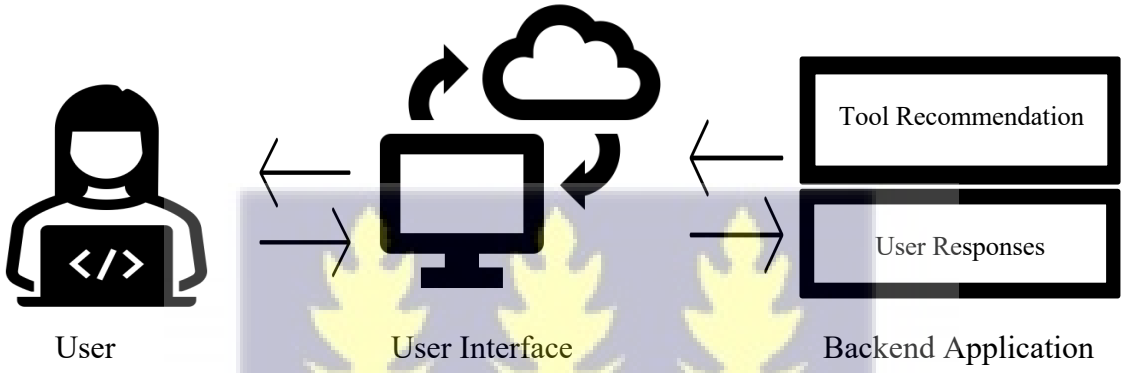

Figure 1: Operation of Test Automation Tool Selector

The system operates by taking in the responses to questions linked to a defined criteria for selecting testing tools, processing the responses and outputting a recommended tool, *Figure 1*. *Figure 2* shows the flow diagram of how the system progresses from user input to tool recommendation, with the WSM at its core module.

In the following subsections, the system design is considered, detailing how these components interact and the rationale behind key design decisions. How the system implements the WSM methodology to arrive at its recommendations, ensuring transparency and traceability in the decision-making process is also explored. By providing a clear, data-driven approach to tool selection, this system aims to save time, reduce potential bias, and ultimately lead to more effective choices in automated testing tool selection.

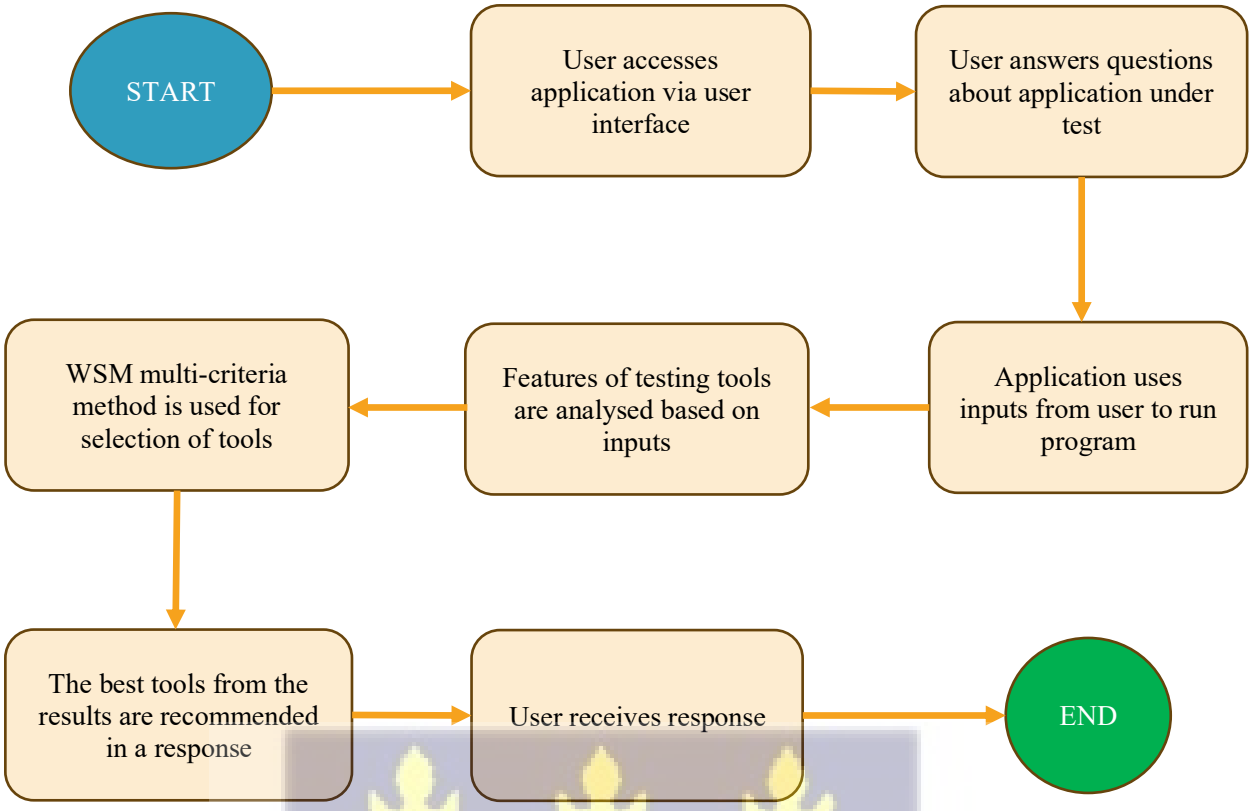

Figure 2: Flow Diagram of System

3.2.2 System Design

The system design for the test automation tool selector is built on the core principle of translating user requirements into structured evaluation criteria and applying the Weighted Sum Model (WSM) to recommend the most suitable automation tool. Unlike traditional subjective or time-intensive decision-making processes, the tool selector provides a systematic approach that ensures all relevant factors are considered and that the final choice reflects user-specific preferences and priorities.

An iterative design cycle is embedded within the development phase to allow for continuous refinement of the application's structure, logic, and evaluation process. This ensures that inconsistencies, gaps, or inefficiencies identified in the design, are addressed before advancing to the next stage. By doing so, the tool strengthens its robustness, accuracy, and adaptability in aligning recommendations with user needs.

In the approach to selecting the best automated tool, two (2) main checks are done during the process of arriving at the best automated testing tool, *Figure 3*. These intermediate checks happen before the actual weighted sum model is applied. Below are the checks.

1. The first check done is to analyse the user's response on the platform to test.
2. The second is to check the type of testing imputed by the user to be done.
3. Based on the first two checks, the user responds to the remaining questions, which are linked to the criteria.

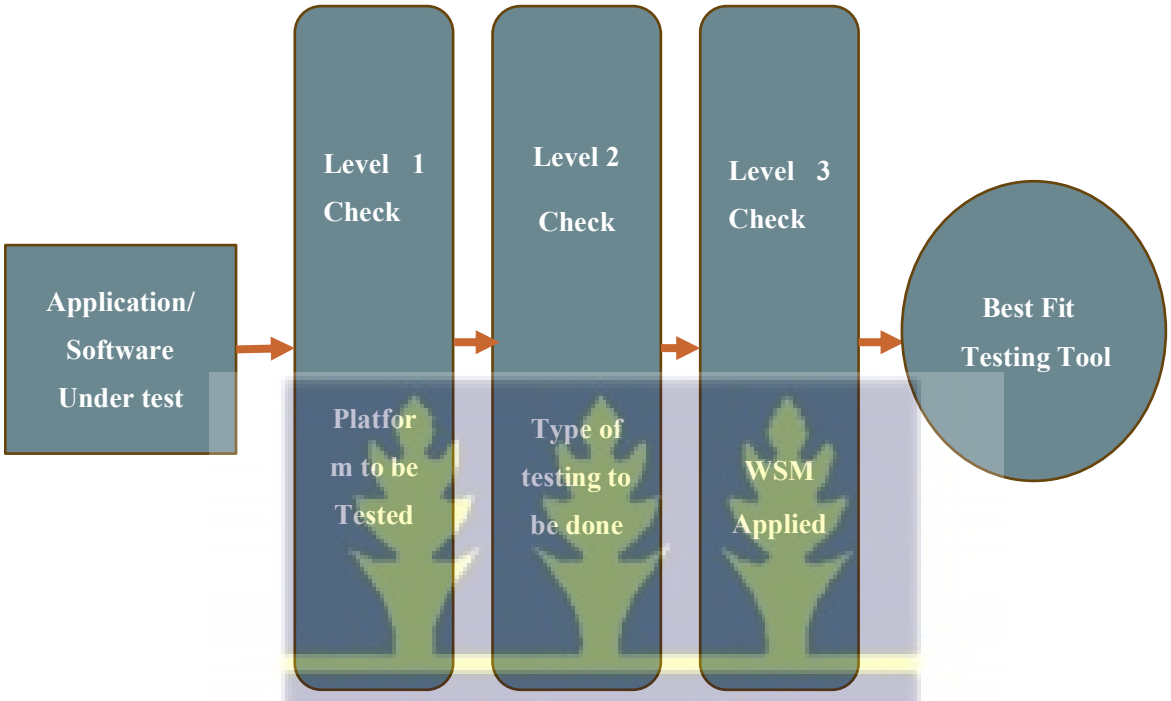

Figure 3: System Design Overview

For most of the related work already done in past research, the platform to test and testing type are usually added as criteria, but in this study, there was a need to separate them. This is because, based on the platform to test and the type of testing being done, the tools can be classified to make the evaluation process more efficient.

3.2.1.3 System Components

The system developed has two main aspects: the user interface (frontend) and the backend that computes the inputs given by the user to give an output. The core functionality of the system is to take inputs or responses from users based on questions asked via the user interface. The system developed consists of several interconnected modules that work together to take responses from the users from initial input to final tool recommendation. The key components of the system are as follows:

1. User Interface: This is the entry point where users interact with the automated testing tool selection system.
2. User Response Collection: In this stage, the user's responses to the questions linked to different criteria and platforms to be tested are taken.
3. Criteria Weight Assignment: Based on user responses, the platform to be tested is selected, and weights are assigned to the different criteria being considered for this study.
4. Database of Testing Tools: Stores information about various testing tools that will be used in the decision-making process for selecting the right testing tool.
5. WSM Calculation Module: For this component of the system, the Weighted Sum Model is applied using the assigned weights.
6. Tool Ranking: Ranks the tools based on the WSM calculations.
7. Best Fit Tool Recommendation: Identifies the most suitable tool based on the ranking.
8. Results Display: Presents the recommended tool to the user.

Figure 4 shows the various component of the system.

3.3 Requirement Analysis and Specifications

This section outlines the requirements and specifications for the automated testing tool selection system. The system aims to assist users in choosing the most appropriate automated testing tool based on their specific needs and preferences, utilising the Weighted Sum Model (WSM) for multi-criteria decision-making method. For any software system to work as expected, there are specific requirements to consider. The requirements outlined in this section provide a clear roadmap for developing the automated testing tool selection system.

By incorporating the specific criteria and the two-step tool classification process (platform selection followed by testing type), the system will delivering highly relevant and tailored recommendations is ensured. The system is more sophisticated compared to others because it can accommodate multiple assessment criteria while incorporating the platform and the type of testing to be done. For this study, to build the test automation tool selector, the following requirements were considered:

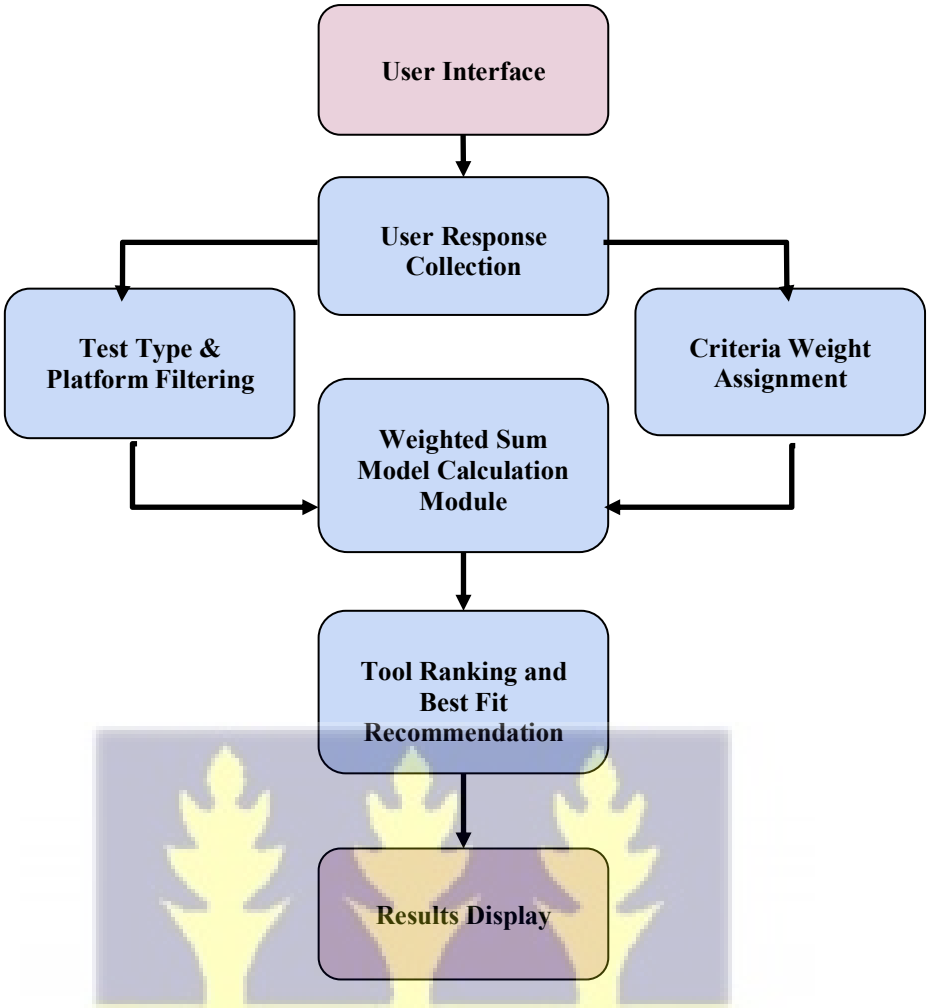

Figure 4: Test Automation Tool Selector Components Diagram

3.3.1 Functional Requirements of the System

Functional requirements refer to the specifications that define the expected behaviour and functionalities of a software system, ensuring it meets user needs [30]. For the system built to function as expected, the following functional requirements must be considered:

1. User Interface
    a. The system must have a user interface to display a series of questions based on the various criteria to gather user preferences and requirements for their software testing.
    b. The user interface must display the right testing tool after the weighted sum model is applied.

2. User Input
    a. The system must prompt users for their testing platform (web, mobile, API).
    b. The system must ask users about their primary type of testing (unit, Performance, Security, etc).
    c. The system must collect user preferences for the tool selection criteria using a 5-point Likert scale.
3. Tool Filtering
    a. The system must store information on the testing tools and their respective performance values.
    b. The system must filter the tool database based on the user's selected platform and testing type.
    c. The system must only consider tools that match both the platform and testing type in subsequent analysis.
4. WSM Analysis
    a. The system must convert user preferences into normalised weights for each criterion.
    b. The system must apply the WSM to calculate scores for each filtered tool.
    c. The system must rank tools based on their calculated scores.
5. Results Presentation
    a. The system shall display a list of recommended tools.

3.3.2 Non-Functional Requirements of the System

Non-functional requirements are a set of specifications that describe a system's operation capabilities and constraints [22]. These are basically the requirements that outline how well a system operates, including things like speed, security, reliability, data integrity [62]. In this work, the following will have to be considered.

1. Performance: The system must provide recommendations within 30 seconds of receiving all user inputs.

2. Usability: The user interface must be intuitive and require no prior training to use. The system should complete the recommendation process in less than 2 minutes for a typical user.
3. Scalability: The system must be able to accommodate at least 100 different testing tools without performance degradation. Adding new evaluation criteria should not require significant system restructuring.
4. Reliability: The system must have an uptime of at least 99.9%. Data integrity must be maintained, with no loss of tool information or user inputs.
5. Maintainability: To ensure seamless updates and enhancements, it is essential that the system architecture is designed with a modular framework.

3.3.3 User Requirements

1. The tool selection process based on the user's input must be efficient
2. There must be personalised recommendations based on specific testing requirements or preference
3. User-friendly interface for inputting preferences and viewing results

For the test automation tool selector system, the main or primary users of the system are listed below.

- Software testers
- Quality assurance engineers
- Project managers
- Product managers
- Development team leads

With these requirements in place, the system can be built to meet high standards of usability, performance, and reliability while fulfilling its primary role of recommending appropriate testing tools and producing results that are contextually relevant and tailored to the user's particular testing scenario.

3.4 Data Gathering and Analysis

This section outlines the methodologies employed for data collection and analysis in the study of selecting the optimal automated testing tool using the Weighted Sum Model (WSM)

multicriteria method. The process involved three aspects: a literature review, an industry survey, and market analysis. This approach ensured a robust and well-rounded set of data for the study.

3.4.1 Data Collection Methods

The various data collection methods are in the following sections.

3.4.1.1 Data Gathering from Literature Review

Drawing on previous research, a thorough literature review was carried out to pinpoint:

- Popular automated testing tools in the market
- Common criteria used for evaluating such tools
- Current trends in automated testing

Databases such as ResearchGate, IEEE Xplore, ACM Digital Library, and Google Scholar, focusing on publications from the last five years to ensure relevance were utilised for the review and data gathering. Key search terms included "selecting the right automation testing tool", "using multi-criteria method for selecting testing tools", "automated testing tools", "software test automation", and "test tool selection criteria".

3.4.1.2 Industry Survey

In addition to the literature review, an online survey targeting industry experts in software testing and quality assurance was conducted in order to gain insights from practising professionals. Experts were selected based on their experience in software testing and automation practices.

They were engaged through structured questionnaires and follow-up discussions to assign relative weights to the evaluation criteria and validate the scoring of candidate tools. This process ensured that the WSM was not only informed by user preferences but also grounded in professional expertise, thereby enhancing the rigor and reliability of the tool selection framework. The details for conducting the survey are listed below.

1. Participants: 30 Software testing professionals and software developers
2. Survey Platform: Google Forms
3. Duration: 4 weeks

4. Industries Represented: Finance, Fintech and Technology
5. Experience Range: 2 to 10+ years in the field

The survey consisted of both closed and open-ended questions, asking participants about:

- Their current automated testing practices
- Their years of experience in software testing
- The criteria they consider when selecting testing tools
- The challenges they face in tool selection
- The industries they represented

3.4.1.3 Market Analysis

Thirdly, market analysis was performed to identify the most relevant automated testing tools currently available. This process involved:

1. Reviewing industry reports like Gartner Magic Quadrant and Forrester Wave
2. User reviews and ratings on platforms like G2 and Capterra were also analysed.
3. Finally, Google search engine together with AI tools like Chat GPT, Mistral, Meta AI, Gemini and Claude were used to search common and relevant testing tools currently available.

3.4.2 Data Analysis

In working on this project, there is a need to take the necessary data to analyse. Firstly, the objective is to use a multi-criteria decision-making method to select the right automated testing tools. This research is mainly focused on software testing and hence the data to conduct this research is targeted at software testers and stakeholders in the software development sector. To use the multi-criteria decision-making method, the first set of information required is the different criteria to be used in the decision-making method. As mentioned earlier, the MCDM method to be used is the weighted sum model. To use this, the set of criteria used for selecting the right testing tools had to be identified.

3.4.2.1 Data from Literature Review Analysis

The thematic analysis method was used to synthesise findings from the literature review. This approach was chosen due to the extensive body of research already available in this field. It

was essential to leverage the insights and findings from previous studies to build upon the established work, ensuring a solid foundation for further exploration and development.

Key themes that emerged included:

1. Importance of tool integration with existing development ecosystems
2. Challenges faced when selecting automation testing tools
3. The key criteria considered when selecting test automation tools.

3.4.2.2 Industry Survey Data Analysis

Survey responses were analysed using both quantitative and qualitative methods using the google form charts and google sheets. *Figures 5,6,7* and *8* are the key findings from the survey conducted during the data gathering process:

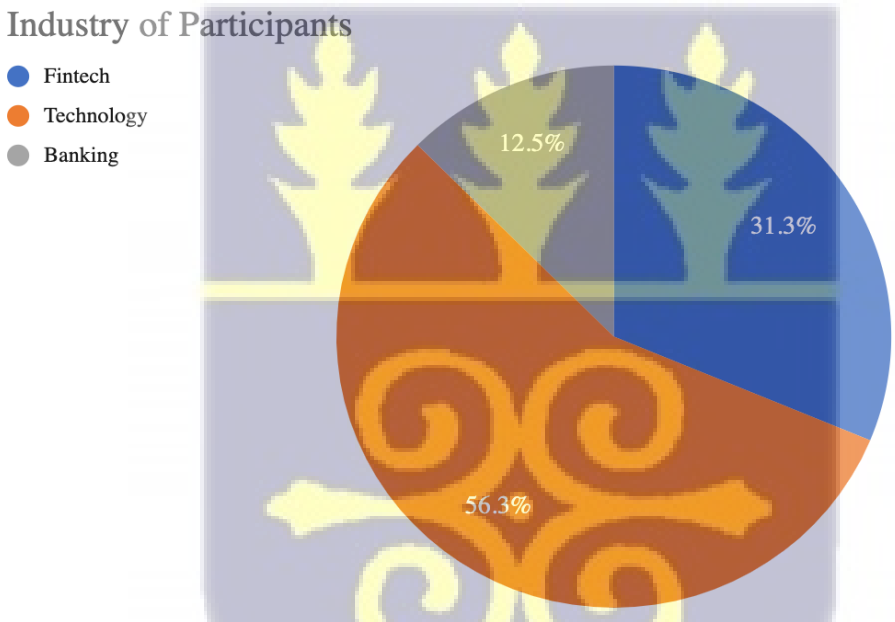

Figure 5: Chart of Industry of the Survey Participants

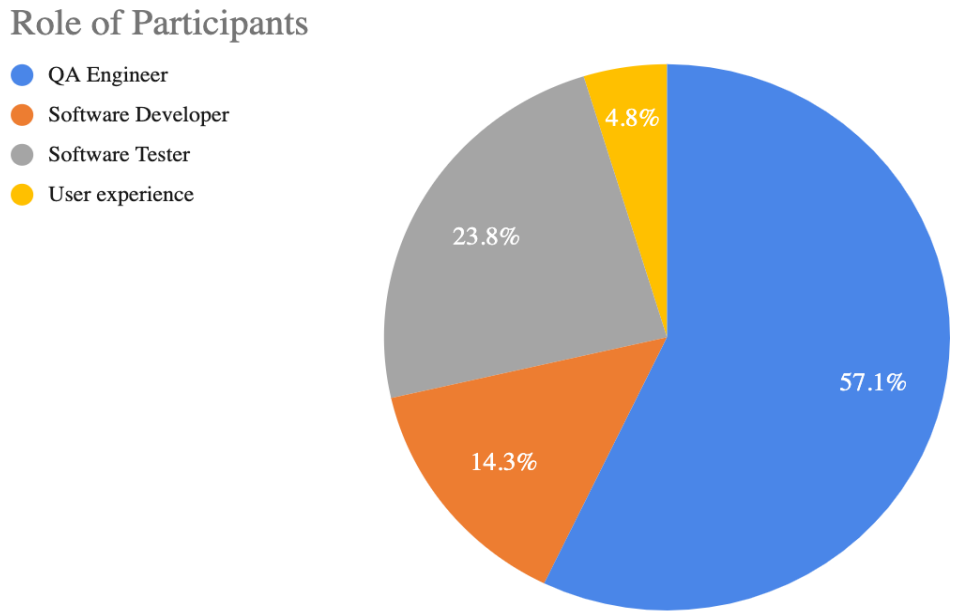

Figure 6: Chart of the Role of the Survey Participants

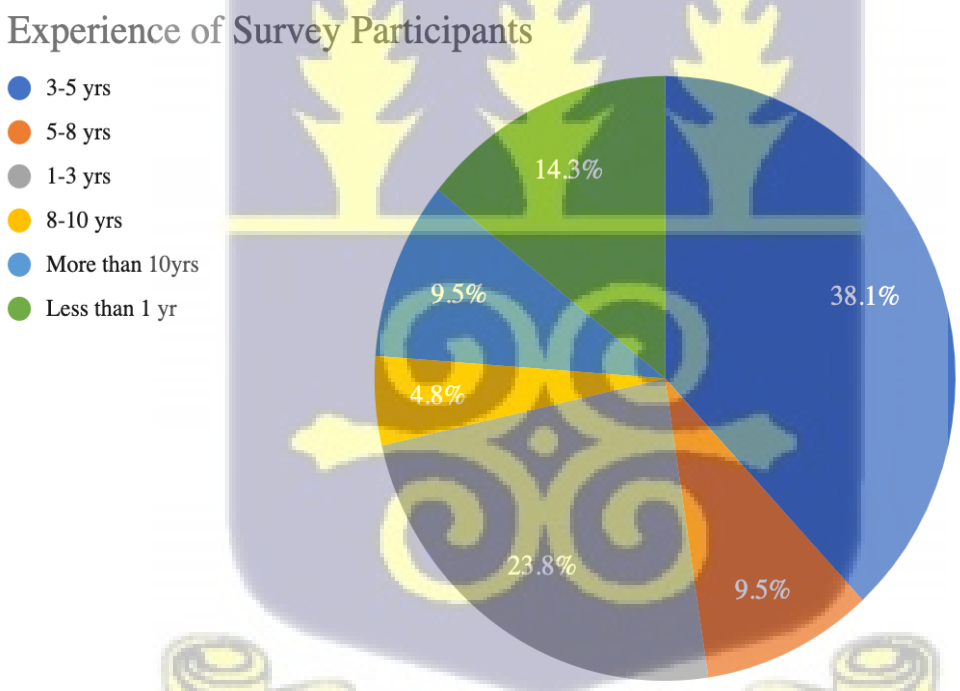

Figure 7: Chart of the Experience of the Survey Participants

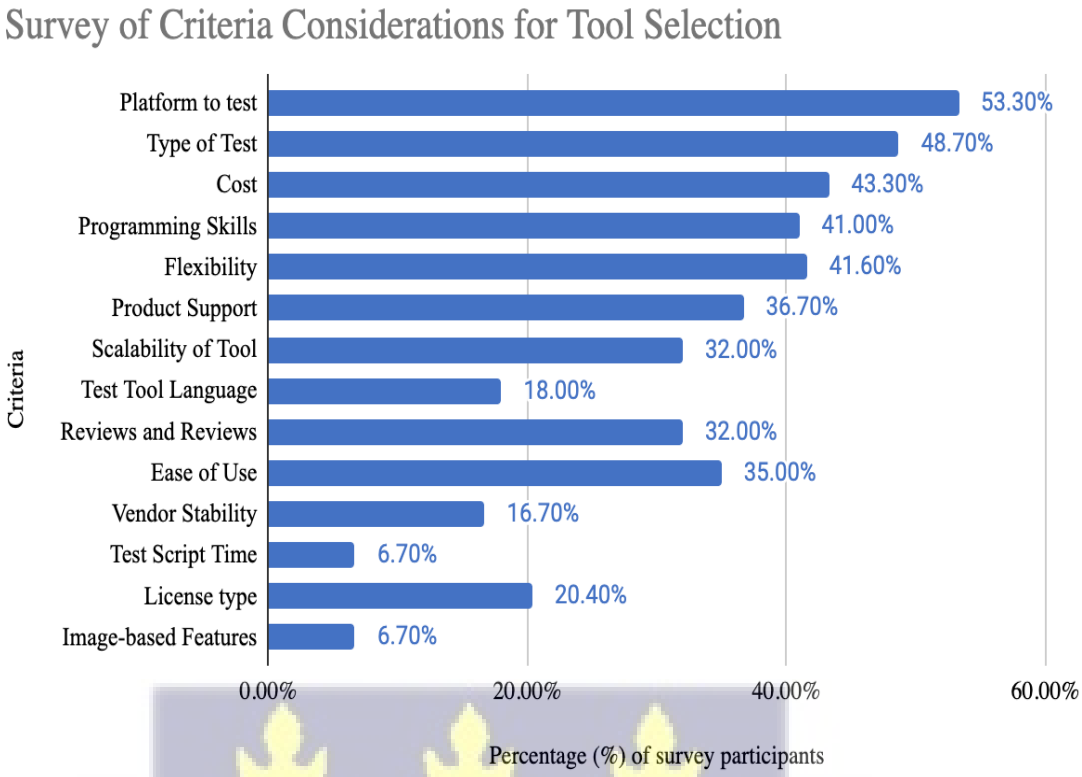

Figure 8: Survey of Criteria Considered for Tool Selection

3.4.2.3 Market Analysis Synthesis

The market analysis data was synthesised to identify the following:

1. Leading tools in different categories (web, mobile, API testing, etc.)
2. Emerging trends in tool features and capabilities
3. Various prices and licensing models currently available on the market for software testing tools

3.4.3 Criteria Selection

Based on the combination of data from the literature review, industry survey, and market analysis, the following criteria were identified as most critical for evaluating automated testing tools:

1. Cost and Licensing: This includes the overall ownership cost, such as the initial purchase price, ongoing licensing fees, and additional expenses for support or training.

2. Programming Skills: The level of programming expertise required to effectively use and maintain the tool. This can range from codeless solutions to those requiring advanced scripting abilities.
3. Flexibility and Customisation: The tool's ability to adapt to specific testing needs, support various testing types, and integrate with existing workflows and tools.
4. Ease of Use: The intuitiveness of the user interface, the learning curve for new users, and the overall user experience of working with the tool.
5. Scalability and Performance: The tool's capability to handle increasing workloads, support large-scale projects, and maintain performance under stress.
6. Vendor Stability and Product Support: The reliability and longevity of the tool's vendor, as well as their vision for future development and support.
7. Reviews and Recommendations: Feedback from other users, industry recognition, and expert recommendations about the tool's effectiveness and reliability.

Figure 9 shows the list of the criteria chosen for this study.

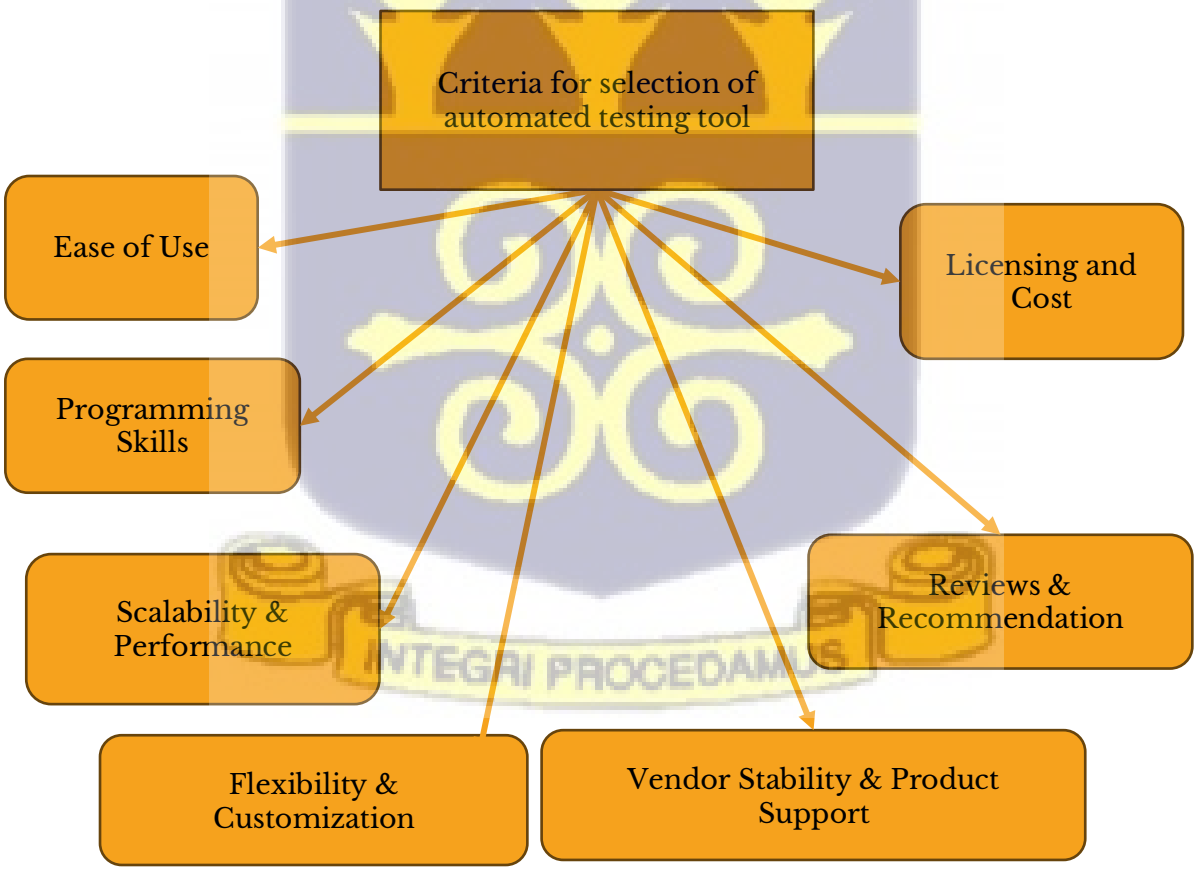

Figure 9: Criteria for WSM Application

These were chosen as a result of the survey. It was seen that the two main considerations made for testing tool selection are platform to test and type of testing. This is why for this work, these two were separated from the criteria-based questions. These two major criteria are used for the tool filtering process. Some of the criteria from the survey responses were combined to come up with the above listed criteria for automation tool selection.

To validate these criteria, a follow-up consultation was done with a panel of five senior QA managers from different industries. Their feedback was incorporated to refine the criteria list and ensure its comprehensiveness and relevance. It is important to note that the significance of these criteria can vary greatly depending on the team's specific needs, project requirements, and organisational constraints.

3.4.4 Testing Types

When it comes to test automation, there are specific types of tests that require test automation. The 5 main types of tests used for this study are:

1. Functional
2. Security
3. Performance
4. Integration
5. Unit testing

These types of tests were specifically used based on the reasons indicated in this section. For any built application, a functional test is what will be used to check if the product is working as expected. Functional testing is a critical process that ensures software operates according to specified requirements by evaluating its functionality through input and output comparisons against expected results. From a functional test, one can determine if an application meets standard quality. For this reason, it must be part of the list of tests to be evaluated during the tool selection process.

The next type of test for this study is security testing. Software security testing is the process to identify whether the security features of software implementation are consistent with the design [67]. Effective security testing requires a blend of expertise and experience, particularly for risk-based approaches that mimic attacker strategies. We live in a digital world now, and

everyone will want to be safe when using any application. From security testing, one can check how safe and secure using an application is.

The next type of test that was considered again is performance testing. Performance testing is a critical process that evaluates a software system's ability to meet specified performance requirements under various conditions. Performance test done to check the non-functional requirements features. This typically involves stress testing and load testing. This tests the system at maximum design load and is done using "virtual" users and tools that simulate real user behaviour. From performance tests, one can determine how persistent an app is.

The integration test was also used as a type of test to be considered in this work. The primary goal is to identify errors related to interfacing and to confirm that the integrated components adhere to the specifications outlined in the Software Requirements Specification (SRS) document. An important part of software development is integration testing, which tests that different modules work together properly. This function is often carried out in conjunction with unit testing, and the approaches are different, such as the big-bang approach, the top-down approach, and the and the bottom-up-to method established [18].

Unit testing is a critical phase in the software development lifecycle, focussing on verifying the functionality of the smallest testable units of code, typically conducted by developers using a white box testing approach. This process allows for early detection of bugs, which can significantly reduce the cost of fixing errors later in development. Unit tests are designed to ensure that each unit operates as intended, and they can be executed in parallel for multiple units [18].

So, from the unit built, tests are done to ensure the unit is working as expected. If it is, then integration tests are done to check how each unit works with other units. If that meets expectations, then the functional testing is performed by testing the whole product to check if it meets standard. If this meets the standard, the next is to check how it performs under stress and under load. If that works well, then one can check how secure the application or software is. Based on the importance and relevance of the above-mentioned types of tests, the evaluation of tools for this research was be done based on these five types of tests: unit, integration, functional, performance and security testing.

3.4.5 Tool Selection

After putting together the types of tests, the next thing to consider is the possible alternatives for the multi-criteria decision-making process. Based on the market analysis and survey results, fourteen (14) automated testing tools were selected with the five (5) types of testing. as the focus. These testing tools were selected with respect to three main types of platforms to test: Application Program Interface (API), mobile, and web applications.

These three platforms to tests were chosen as the first level of check during the tool selection process. This is mainly because most software built is now web-based or accessed via mobile applications. APIs are used to interface between most web apps; hence the choice of using these three options. The second level of check is the type of testing that will be performed on the selected platform to test.

The set of testing tools was limited to fourteen (14) for this study because each tool had to be evaluated to determine its performance rating. After investigating automated testing tools on the market via survey, Google searches, and testing tools, the following were the reasons for the choice of tool selection:

1. Market share and popularity
2. Relevance to modern software development practices
3. Diversity in features and target use cases

3.4.5.1 Tool List

Each tool has specific features that play a key role in the evaluation process. Below are details of the fourteen (14) tools selected for this research, and *Table 3* shows the summary of their main functions and features. The tools selected were based on survey responses and market research to get commonly used tools and relevant tools for the various types of testing chosen for this study.

The list of testing tools can always be updated in the database once the various performance ratings are attached accordingly.

1. **Selenium**

   Pricing: Open-source and free.

   Test Platform: Web.

Type of Testing: Functional, Integration, Regression.

Key Features:

- Supports a variety of programming languages, including Java and Python.
- Offers broad compatibility with browsers like Chrome, Firefox, and Safari.
- Can integrate with different testing frameworks and CI/CD tools.

2. **Cypress**

Pricing: Open-source version available; paid plans for advanced features.

Test Platform: Web.

Type of Testing: Functional, End-to-End.

Key Features:

- Fast test execution with real-time reloads.
- Developer-friendly interface with built-in debugging tools.
- Supports time-travel debugging for easier test analysis.

3. **Appium**

Pricing: Open-source and free.

Test Platform: Mobile (Android and iOS).

Type of Testing: Functional, Integration.

Key Features:

- Supports hybrid, native, and mobile web applications.
- Utilises the WebDriver protocol, allowing for cross-platform testing.
- Flexible and allows tests to be written in multiple programming languages (Java, Ruby, Python, etc.).

4. **Espresso**

Pricing: Free and open source (part of Android SDK).

Test Platform: Mobile (Android).

Type of Testing: Functional, Unit.

Key Features:

- Integrated with Android Studio, simplifying setup and usage.

- Designed for quick and reliable UI tests.
- Offers advanced synchronisation features for handling UI components.

5. **XCUITest**

   Pricing: Free and open source (part of Xcode).

   Test Platform: Mobile (iOS).

   Type of Testing: Functional, Unit.

   Key Features:

   - Seamlessly integrates with Xcode for iOS app development.
   - Supports both unit and UI testing.
   - Provides real-time feedback and extensive reporting capabilities.

6. **Test Complete**

   Pricing: Commercial; free trial available (pricing varies by license type).

   Test Platform: Web, Mobile, and Desktop.

   Type of Testing: Functional, Regression, Integration.

   Key Features:

   - Supports multiple scripting languages (JavaScript, Python, etc.).
   - Powerful object recognition engine for intuitive test creation.
   - Offers keyword-driven testing and detailed reporting.

7. **Postman**

   Pricing: Free tier available; paid plans for advanced features.

   Test Platform: API.

   Type of Testing: Functional, Integration.

   Key Features:

   - User-friendly interface for building and testing APIs.
   - Supports automated testing using collections and pre-request scripts.
   - Offers collaboration features for team environments.

8. **SoapUI**

   Pricing: Open-source version available; commercial version (SoapUI Pro) is paid.

Test Platform: API (SOAP and REST).

Type of Testing: Functional, Security, Performance.

Key Features:

- Advanced testing capabilities, including security and load testing.
- Supports Groovy scripting for complex test scenarios.
- Comprehensive environment for API testing with built-in reporting.

9. JMeter

Pricing: Open-source and free.

Test Platform: Performance testing (Web applications).

Type of Testing: Performance, Load.

Key Features:

- Supports various protocols (HTTP, FTP, JDBC, etc.).
- Extensive reporting and visualisation capabilities.
- Enables distributed testing across multiple systems.

10. REST Assured

Pricing: Open-source and free.

Test Platform: API (REST).

Type of Testing: Functional, Integration.

Key Features:

- Simple syntax for writing tests in Java.
- Easy validation of responses and request specifications.
- Integrates well with Java testing frameworks like JUnit.

11. Puppeteer

Pricing: Open-source and free.

Test Platform: Web.

Type of Testing: Functional, Integration, Performance.

Key Features:

- Controls Chrome or Chromium over the DevTools Protocol.
- Supports automated testing, web scraping, and generating PDFs.
- Simple and flexible API for browser automation tasks.

12. TestCafe

Pricing: Free and open source; paid options for support.

Test Platform: Web.

Type of Testing: Functional, End-to-End.

Relevant Features:

- No browser plugins required; tests run directly in the browser.
- Supports asynchronous testing and easy syntax.
- Built-in parallel test execution for faster testing.

13. Burp Suite

Pricing: Free community edition; paid professional version.

Test Platform: Web (Security).

Type of Testing: Security.

Key Features:

- Comprehensive tools for web application security testing.
- Features for crawling, scanning, and vulnerability assessment.
- Supports extensive plugins for added functionality.

14. OWASP ZAP (Zed Attack Proxy)

Pricing: Open-source and free.

Test Platform: Web (Security).

Type of Testing: Security.

Key Features:

- User-friendly interface suitable for beginners and professionals.
- Supports both automated and manual testing strategies.
- Strong community support and frequent updates.

3.4.5.2 Tool Evaluation

Each of the selected tools was evaluated against the criteria. The evaluation process involved:

1. Review of official documentation and user guides
2. Analysis of user reviews and expert opinions
3. Pairwise comparison
4. Hands-on testing of some of the tools in a controlled environment
5. Consultation with vendor representatives for clarification on certain features

For each criterion, tools were rated on a scale of 1-10, where 1 represents poor performance and 10 represents excellent performance. The scores were then normalised for purposes of the study. To ensure objectivity, each tool was evaluated independently by three industry experts to determine their performance ratings against the criteria selected for this research. *Table 4* gives the details of each tool against the 7 criteria used for this study and the performance rating of the tools. The performance ratings of the tools were based on features of the tools and industry experts' ratings from surveys conducted.

Table 3: Tool classification based on Platform to test and Testing Types

Testing tools	**Platform to test**	**Testing Requirement**
Appium	Mobile Application	Functional, Performance, Integration
Espresso	Mobile Application	Functional, Performance, Unit
XCUITest	Mobile Application	Functional, Performance, Unit
TestComplete	Web Application, Mobile Application, API Services	Functional, Security, Performance, Unit, Integration
Postman	API Services	Functional, Performance, Integration
SoapUI	API Services	Functional, Security, Performance
JMeter	API Services	Functional, Performance
REST Assured	Web Application, API Services	Functional, Performance, Integration

Selenium	Web Application	Functional, Integration, Unit
Cypress	Web Application	Functional, Integration
Puppeteer	Web Application	Functional, Performance
TestCafe	Web Application	Functional
Burp Suite	Web Application	Security
OWASP ZAP (Zed Attack Proxy)	Web Application	Security

3.5 Theoretical framework and assumptions

Based on the approach for this research, choosing automated testing tools is based on several important theoretical frameworks, and it is important to comprehend the assumptions made. The basic concepts and their rationale are described in this section.

3.5.1 Multi-Criteria Decision Making (MCDM) Methods

The core theoretical framework underpinning this research is Multi-Criteria Decision Making (MCDM). Multi-Criteria Decision Making (MCDM) is a vital process in operations research that aids in selecting the best feasible solution among alternatives based on multiple, often conflicting criteria. According to Hwang and Yoon, MCDM encompasses two main categories: multi-objective decision-making (MODM) and multi-attribute decision-making (MADM), each employing various methods tailored to specific decision contexts [68].

3.5.1.1 Multi-Objective Decision Making (MODM)

The values of the decision variables in MODM techniques are determined in a continuous or integer domain with either an infinitive or a large number of alternative choices [69]. Based on the principles underlying how MODM works, this method cannot be used for this research as the number of set of choices (automated testing tools) that will be used for the selection process is finite.

Table 4: Performance Ratings of Selected Tools Against the Criteria

Tools/ Criteria	Ease of Use	Programming Skills	Scalability and Performance	Flexibility and Customization	Cost and Licensing	Vendor Stability and Product Support	Reviews And Recommendations
Appium	0.063	0.063	0.091	0.078	0.078	0.078	0.065
Espresso	0.104	0.079	0.055	0.063	0.078	0.078	0.048
XCUITest	0.104	0.079	0.055	0.063	0.078	0.078	0.048
TestComplete	0.042	0.063	0.091	0.078	0.047	0.063	0.081
Postman	0.104	0.079	0.055	0.078	0.063	0.063	0.081
SoapUI	0.104	0.079	0.091	0.063	0.063	0.078	0.065
JMeter	0.063	0.079	0.091	0.063	0.078	0.078	0.081
REST Assured	0.021	0.063	0.055	0.063	0.078	0.063	0.065
Selenium	0.063	0.063	0.091	0.063	0.078	0.078	0.081
Cypress	0.104	0.079	0.091	0.078	0.078	0.063	0.081
Puppeteer	0.042	0.079	0.055	0.078	0.078	0.063	0.065
TestCafe	0.104	0.079	0.073	0.078	0.078	0.063	0.081
Burp Suite	0.021	0.048	0.055	0.078	0.047	0.078	0.081
OWASP ZAP	0.063	0.063	0.055	0.078	0.078	0.078	0.081

3.5.1.1 Multi-Attribute Decision Making (MADM)

MADM methods, on the other hand, are used for making preference decisions over the available alternatives, which are characterised by multiple attributes. MADM are generally discrete, unlike MODM, with a limited number of prespecified alternatives [69]. Based on how the MADM works, this method can be employed for this study. Both the set of alternatives (the testing tools) and a set of different attributes (criteria) for the selection process for this study are finite. An MADM or MCDM problem with finite possibilities can be expressed in a matrix format as shown in Formula (1).

Multi-Attribute (Criteria) Decision-Making Matrix Problem [70]

	Criteria				
	C_1	C_2	C_3	...	C_n
Alternatives	$(w_1$	w_2	w_3	...	$w_n)$
A_1	a_{11}	a_{12}	a_{13}	...	a_{1n}
A_2	a_{21}	a_{22}	a_{23}	...	a_{2n}
.	.	.	.	.	.
.	.	.	.	.	.
A_m	a_{m1}	a_{m2}	a_{m3}	...	a_{mn}

(1)

In the above problem representation:

1. A_i $(i = 1...m)$ represents the different available options or alternatives in the case of this study, the 14 testing tools that will be used for evaluation.
2. C_j $(j = 1...n)$ represents the multi-attributes for the MADM problem. For this study, the selected seven criteria from which users will be questioned for their preference make up the list of attributes.

3. w_j represents the criteria weights. For this study, these weights will be assigned based on the user's preference per the criteria in question.
4. Each alternative has a performance rating, a_{ij} per the criteria being evaluated against. Thus, the rating of alternative i with respect to criterion j.

MCDM in general provides a structured approach to problems involving multiple criteria. As MCDM applications grow, it is essential to choose appropriate methods that align with the problem's characteristics and stakeholder preferences, ensuring a transparent and systematic decision-making process [30]. In the context of automated testing tool selection, MCDM is particularly relevant as it allows for the consideration of various tool attributes simultaneously.

Multi-criteria decision analysis (MCDA) is a structured approach for decision-making that incorporates various criteria, such as costs and environmental impacts, to identify the best alternatives among multiple options. The effectiveness of MCDA methods varies, as different methods can yield different rankings for the same input data, necessitating careful selection based on the desired outcome.

3.5.2 Weighted Sum Model (WSM)

The WSM, a subset of MCDM techniques, aligns well with the complexity of tool selection, allowing for the incorporation of both objective measures and subjective preferences. This model assumes that the overall value of an alternative (thus, a testing tool for this work) can be expressed as the weighted sum of its performance across various criteria.

The WSM involves the following steps:

1. Identification of alternatives and criteria
2. Assignment of weights to each criterion
3. Scoring of alternatives against each criterion
4. Calculation of weighted sum for each alternative
5. Ranking of alternatives based on their total scores

The application of WSM to automated testing tool selection offers the below listed advantages:

1. It provides a structured approach to evaluating tools across multiple dimensions
2. It allows for the incorporation of organisation-specific priorities through criterion weighting

3. It facilitates transparent and justifiable decision-making
4. It can be easily adapted as new criteria or tools emerge in the market

The WSM is particularly suitable for the complex task of automated testing tool selection as it allows for the integration of multiple criteria and user preferences into a single, quantifiable decision-making process. By applying the WSM to the domain of automated testing tool selection, this research aims to develop a comprehensive, adaptable framework that can guide organisations in making informed, objective decisions aligned with their specific testing needs and organisational contexts.

3.5.2.2 Utility Theory in Weight Determination

Underlying the WSM is utility theory, which provides a framework for quantifying subjective preferences. It is assumed that users can express their preferences in a way that reflects their underlying utility function for each criterion. This allows the treatment of the normalised weights as representations of the marginal rate of substitution between criteria in the user's decision-making process.

3.5.3 Assumptions

1. Preference Articulation: Users can accurately articulate their preferences based on the responses to the questions that have been carefully framed to address each of the criteria evaluated in this study. This assumes a level of self-awareness and the ability to quantify subjective feelings.
2. Consistency Over Time: User preferences remain relatively stable over the decision-making period. However, it is acknowledged that preferences may evolve as users gain more information about testing tools.
3. Independence of Criteria: The preference for one criterion does not significantly influence the preference for another. While some interdependence may exist in reality, this simplification is necessary for the WSM.
4. Completeness of Information: Users have sufficient information about each criterion to make informed preference judgements.

3.6 Design and development process

This section outlines the design and development process of the automated testing tool selection system. The system is designed to guide users through a decision-making process, starting with high-level questions about their testing needs and then proceeding to a detailed evaluation of tool criteria using the Weighted Sum Model (WSM) to recommend the appropriate testing tools based on their responses.

3.6.1 Design Overview

The system consists of three main components:

1. User Interface: This component collects user inputs through a series of questions.

2. Tools Filtering Module: This module uses initial responses to narrow down the tool options.

3. WSM Analysis Module: This aspect of the application applies the WSM to the filtered tools based on user-weighted criteria.

3.6.2 User Interface Design

The interface begins with an instruction on what the automation selection is about and how to get it to work. Two crucial questions to narrow down the tool selection begins the set of questions:

1. What platform are you testing? (Options: Web, Mobile, API)

2. What type of testing are you primarily performing? (Options: Functional, Performance, Security, Unit, Integration)

Following the initial questions, users are presented with a series of questions related to each criterion. Users indicate their response based on the options available in the responses. The responses in the drop-down menu use a 5-point Likert scale. Each response is tied to a weight that is used for the WSM calculation.

Sample question: How important is user-friendliness, GUI and ease of use for your team?

Drop down menu responses:

- 1- Not Important
- 2 - Not very important
- 3 - Neutral

- 4 - Important
- 5 - Very Important

The questions are presented as drop-down menus to ensure clear, unambiguous responses. Using any other format like allowing the users to enter their own responses would have eliminated uniformity and increased errors. With the limitation to select from the drop-down menu, the responses from users are easily determined and the weights can be assigned based on their responses.

3.6.3 Tools Filtering Module

3.6.3.1 Tools Database

A comprehensive database of automated testing tools was created, including the following information for each tool:

- Name
- Supported platforms
- Types of testing supported
- Detailed specifications for each evaluation criterion

3.6.3.2 Tools Filtering Process

Based on the user's responses to the initial two questions, the system filters the tool database:

1. Platform filter: Eliminates tools that don't support the selected platform.

2. Testing type filter: Further narrows down the list to tools that specialise in or strongly support the selected testing type.

The main purpose of this filtering is to ensure that only relevant tools are considered in the subsequent WSM analysis. Without the filtering of tools based on the platform to test and type of testing being done, the comparison of the tools and their evaluation will vary. By doing the filtering, the set of tools that do not fall under the type of testing listed will be eliminated. Also, the primary test being done limits the comparison of tools. Only tools that have features to perform the type of testing indicated by the user will be evaluated for the best to be recommended. *Figure 10* shows the process of tool filtering and selection.

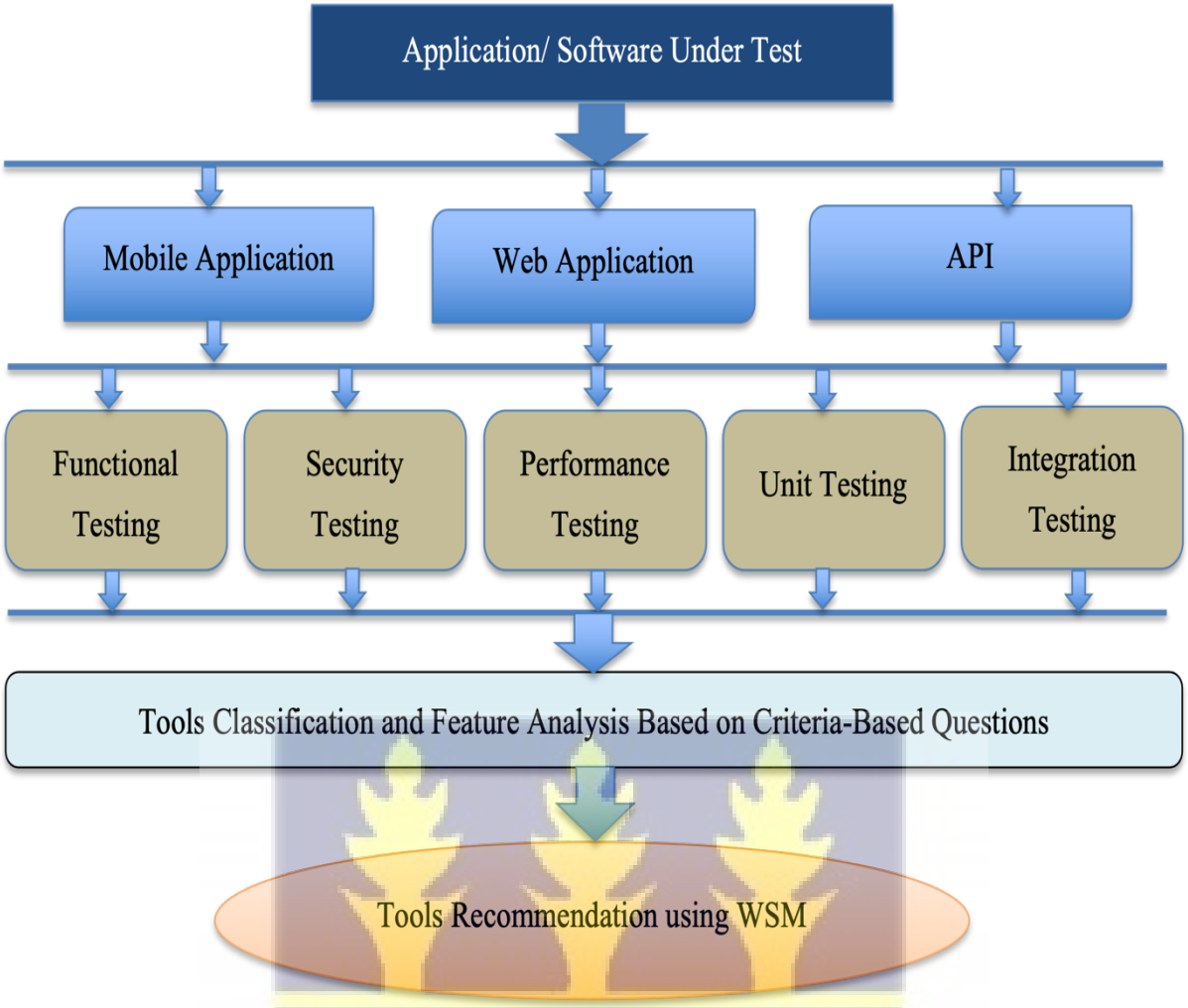

Figure 10: Tools Filtering Process

3.6.2.3 WSM Analysis Module

The system converts user responses from the Likert scale into numerical weights (0.2 -1) for each criterion and the following steps are followed. This scale was used in order for us to have a systematic way of assigning the weights based on the user's responses. All the questions asked had their responses linked to the 5-point Likert scale making the calculator of weights simpler and easier for the WSM model application. Each tool in the filtered list is evaluated against the criteria. This evaluation is based on pre-assigned scores in the tool database as indicated in *Table 4*, which were determined through expert analysis, testing and extensive research of each tool.

From the responses of the user, each of the criteria weights is assigned and the respective performance rating of the filtered tools is multiplied for each alternative using the WSM

formula. The alternative with the highest score is recommended as the right testing tool and the next highest is also shared. Three best tools are displayed after responses are submitted.

3.6.2.4 Weight Determination

The critical aspect of the weighted sum model in the context of this study is how the user preferences to the criteria-based questions are used to derive and apply the weights (w_j) for calculating the weighted sum score. To translate the user preferences into quantitative weights for the WSM, each of the user responses is used to represent the level of importance of the specific criteria in question.

The process is as follows:

1. Users are asked to rate the importance of each criterion based on the question asked. For instance, when a user is asked, “How important is community support and user forums for troubleshooting?" a response of "Very important" gives a 5 on a predefined scale of 1-5 which is converted to 0.2 to 1 in the implementation code.
2. These weights generated in a similar way for each of the criteria questions the user responds to are normalised in order to have consistency in the application of the WSM.
3. Based on the weights determined from the user preferences, the weight per the criteria is multiplied by the performance rate of each tool, and the ones with the highest score are recommended as the best testing tool for the user for that specific project.

After weights are determined and assigned based on the user's response, the filtered tools based on the platform to test, and the type of testing are evaluated and the tools with the highest scores are recommended as the best automated testing tools based on the user's requirements.

3.6.3 Development Process

3.6.3.1 Technology Stack

The main technology stack or tools used for this work is Python. In order to have a more interactive interface, a web application version of the study was also developed. This was done with HTML and CSS. Details are indicated below:

Version 1 : Python was used for both the user interface, computation and presentation of results.

Version 2: Web application

Frontend: HTML 5 and CSS for a responsive and interactive user interface

Backend: JavaScript linked with a public AI model (Mistral AI) was used for the computation of the results.

Infrastructure: Vercel cloud service

The development process followed these stages:

1. Requirements Gathering: Based on the research aims and goals, the information needed for the system, that is, tools database, surveys and questionnaire were performed in this stage. The requirements of the system were compiled in this stage.
2. Development: This process involved designing the architecture of the application, the development of features, testing with continuous integration practices based on feedback received from testers and stakeholders.
3. Deployment: The system was developed and deployed on Python Application for the version 1 and for the version 2, it was deployed on a cloud platform called Vercel for scalability and accessibility

3.6.3.2 Validation and Testing

Usability tests were conducted by people with varying expertise levels in software testing and others with no software testing background. Feedback was used to refine the user interface and question framing. Load and stress testing were also done to ensure the system can handle multiple concurrent users without significant latency.

To validate the system's recommendations, the following steps were followed:

1. Related test scenarios with predefined "ideal" tools were created.
2. Testing experts were made to use the system for these scenarios.
3. The system recommendations were then compared with expert expectations.
4. The experts were asked to rate the system based on the questions in Appendix A.

In conclusion, the design and development process of this automated testing tool selection system aimed to create a user-friendly, accurate, and valuable resource for software testing professionals. By combining initial filtering questions with a WSM-based detailed analysis, the system provides personalised tool recommendations based on specific user needs and preferences.

3.7 Modelling and Simulation of the System

This section details the modelling and simulation processes used in the development and validation of the automated testing tool selection system. The core of this system is based on the Weighted Sum Model (WSM), implemented in Python. Details of the implementation code are explained in the design implementation of Chapter Four (4).

3.7.1 Mathematical Model

Weighted Sum Model is expressed mathematically as

$$A_{WSM\ score} = \max_{i} \sum_{j=1}^{n} a_{ij} w_j, \quad for\ i = 1, \qquad (2)\ [70]$$

m represents alternatives

n represents decision criteria

w_j denotes the relative weight of importance of the criterion, C_j

a_{ij} is the performance value of alternative A_i when evaluated in terms of criterion C_j

In the context of this study, the WSM can be formally expressed as:

$$A_{(WSM\ score)} = (w_U * a_{iU}) + (w_P * a_{iP}) + (w_S * a_{iS}) + (w_F * a_{iF}) + (w_C * a_{iC}) + (w_V * a_{iV}) + (w_R * a_{iR}) \qquad (3)$$

Where: Performance of tool is a_i

Weight of criteria is w_j

The different criteria for this work are denoted by the following:

Ease of Use, U

Programming Skills, P

Scalability and Performance, S

Flexibility and Customisation, F

Licensing and Cost, C

Vendor Stability and Product Support, V

Reviews and Recommendation, R

3.7.2 Simulation of Model

The simulation incorporates three (3) of the testing tools chosen for this study based on the Functional testing and Mobile application as the platform of testing. For the simulation of the system, Microsoft Excel was used. The automated testing tools used were: Appium, Espresso and TestComplete. The details of the simulation process are as follow:

1. Weights were assigned to each criterion as assumptions of the user's responses. Below is the table with details.

Table 5: Simulation - Criteria Weights and Testing Requirements

Testing Requirement	Functional Testing
Platform to test	Mobile Application
Criteria	**Weight**
Ease of Use	0.152
Programming Skills	0.152
Scalability and Performance	0.152
Flexibility and Customization	0.152
Cost and Licensing	0.152
Vendor Stability and Product Support	0.091
Reviews And Recommendations	0.152

2. Each tool is characterised by its platform compatibility, testing capabilities, and performance ratings across all criteria. Each of the tool's performance rating against the criteria was indicated as per the below.

Table 6: Simulation - Tools Performance Ratings per Criterion

Criteria	Appium	Espresso	TestComplete
Ease of Use	0.063	0.104	0.042
Programming Skills	0.063	0.079	0.063
Scalability and Performance	0.091	0.055	0.091
Flexibility and Customization	0.078	0.063	0.078
Cost and Licensing	0.078	0.078	0.047
Vendor Stability and Product Support	0.078	0.078	0.063
Reviews and Recommendations	0.065	0.048	0.081

The WSM was used to calculate the score of each of the tools using the mathematical formula indicated in section *3.7.1 above.*

$$A_{(WSM\ score)} = (w_U * a_{iU}) + (w_P * a_{iP}) + (w_S * a_{iS}) + (w_F * a_{iF}) + (w_C * a_{iC}) + (w_V * a_{iV}) + (w_R * a_{iR})$$

Calculating the score of each tool:

Tool 1- Appium

$Appium_{(WSM\ score)}$ = (1.175*0.063) + (0.097*0.063) + (0.171*0.091) + (0.09*0.078) + (0.198*0.078)+ (0.168*0.078) + (0.098*0.065) = **0.0748**

Tool 2 - Espresso

$Espresso_{(WSM\ score)}$ = (1.175*0.104) + (0.097*0.079) + (0.171*0.055) + (0.09*0.063) + (0.198*0.078)+ (0.168*0.078) + (0.098*0.048) = **0.0744**

Tool 3 - TestComplete

$TestComplete_{(WSM\ score)}$ = (1.175*0.042) + (0.097*0.063) + (0.171*0.091) + (0.09*0.078) + (0.198*047)+ (0.168*0.063) + (0.098*0.081) = **0.0641**

From the results of the calculations, Appium had the highest score indicating that it ranks first as the best tool fit for functional testing on mobile applications. By this same process, each tool

filtered based on the user's response will be checked and the best first three based on the highest scores will be recommended as the best testing tools.

3.8 Development Tools and Material Requirements

The key tools used in this study are the software development tools.

1. The primary programming language used was Python.
2. The Integrated Development Environment (IDE) used PyCharm and Visual Studio Code
3. For the web version of the tool used for this study, HTML 5, CSS and JavaScript were connected to Mistral AI.
4. Python and Microsoft Excel were used analysing the data for this study.

These were the choices made for this study mainly because of the analytical functionality Python has. The code is simple to update and scale up for a programming language like Python. Mistral was chosen because it is known for its cost-effectiveness, speed, and reliability, excelling in translation, summarization, and sentiment analysis. It is also because the Mistral model is trained specifically for code tasks. The choice of IDE was as a result of the programming language chosen.

CHAPTER 4

DESIGN IMPLEMENTATION AND TESTING

4.1 Introduction

This chapter outlines the design, implementation, and testing of the automated testing tool selection system developed using Weighted Sum Model (WSM) decision framework. Building upon the background research and conceptual framework established in the previous chapters, this section delves into the practical development of the proposed WSM-based system. The primary objective of this work is to create a tool that can autonomously evaluate and select the most appropriate automated testing tool using an MCDM method. This tool implementation is based on a defined set of criteria and responses to questions linked to this set of criteria.

In the sections that follow, the specific design of the WSM-based automated testing tool selection system will be detailed, including the key criteria used. The assignment of weights, and the integration of the decision model with functional software components will be discussed. Finally, the testing procedures and results is presented, demonstrating how the developed tool was validated against real-world scenarios and performance requirements. This will provide evidence of the system's efficacy in automating the complex task of testing tool selection using the Weighted Sum Model.

4.2 The Design Framework

The design framework for the automated testing tool selection system is centred around a multi-criteria decision analysis method called the Weighted Sum Model (WSM). The WSM methodology was chosen as the underlying decision model given its ability to handle multiple factors by assigning weights to each criterion when evaluating alternatives. This approach was also chosen to handle the evaluation of testing tools based on a variety of factors that may be conflicting or have different levels of importance. The core components of the design framework are:

1. User Input Collection**:** The system prompts the user to provide responses to a series of questions (Appendix B) covering important criteria such as the testing platform, testing requirements, ease of use, programming skills, scalability needs, licensing preferences, and more. This user input is then used to drive the decision-making process.

2. Tool Filtering: Based on the user's responses about the target platform and testing requirements, the system filters the available testing tools down to a subset that are relevant and compatible.
3. Criteria Weighting: The design includes a predefined set of criteria weights that map the user's responses to numerical values. For example, if the user indicates that "Ease of Use" is "Very Important", it is assigned a weight of 1.0, while "Not Important" would be 0.2. These weights are used to calculate the overall scores for each testing tool.
4. Weighted Score Calculation**:** For the filtered tools, the system calculates a weighted sum score based on the user's preferences and the predefined ratings for each tool across the various criteria.
5. Top Tool Recommendations: The final step is to sort the tested tools by their calculated scores and present the top 3 recommendations to the user.

This design framework allows the system to systematically evaluate multiple testing tools against the user's specific needs and preferences, providing a data-driven recommendation.

The corresponding data structures and algorithms of the key components are as follows:

1. User Input Collection:
    - Data Structures:
        - `valid_options` - A list of valid responses for each user input prompt
    - Algorithms:
        - `get_valid_input()` - A function that prompts the user for input and validates it against the `valid_options` list
2. Criteria Weighting:
    - Data Structures:
        - `criteria_weights` - A dictionary that maps user responses to numerical weights for each criterion
    - Algorithms:

- `get_user_weights()` - A function that applies the user's responses to the `criteria_weights` dictionary to calculate the final user weights
- `normalise_weights()` - A function that normalises the user weights to ensure they sum up to 1

3. Tool Filtering:
   - Data Structures:
     - `tools` - A list of dictionaries representing the available testing tools, with fields for platform, testing requirements, and other metadata
   - Algorithms:
     - `filter_tools()` - A function that takes the user's platform and testing requirements, and filters the `tools` list accordingly
4. Weighted Score Calculation:
   - Data Structures:
     - `ratings` - A dictionary that stores the predefined ratings for each tool across the various criteria
   - Algorithms:
     - `calculate_tool_scores()` - A function that computes the weighted sum score for each filtered testing tool, using the user weights and `ratings` data
5. Top Tool Recommendations:
   - Data Structures:
     - `tool_scores` - A dictionary that stores the calculated scores for each filtered tool
   - Algorithms:
     - `get_top_tools()` - A function that sorts the `tool_scores` dictionary and returns the top 3 recommendations

4.3 Implementation of the design

This section outlines the overall architecture of the system, detailing the key components and their interactions with one another. Two versions were considered for the design implementation. The first was implementing the weighted sum model MCDM via a Python program. The second method was to develop a web application linked to an existing AI model - Mistral AI. These two approaches were considered to give us a broader view of outcomes for an effective analysis of the method chosen.

MCDM methods work with a set of criteria and alternatives. For this study, the fourteen (14) selected tools were chosen, based on 5 main types of testing. The implementation of the automated testing tool selection system closely follows the design framework outlined in the previous section. From the preferences of the users indicated in response to the questions asked, WSM is used to calculate scores for each tool filtered. The best tools are recommended based on the highest scores.

4.3.1 Python Code Implementation

Python Code implemented for this study is in Appendix C. Explained design implementation is as follows:

1. The implementation of the code starts with the user input collection. This is where the user is presented with questions as per the sample code below.

```
def get_user_input():
    print("This is an application that recommends testing tools based on your responses. Respond to the questions below to see the recommended tools for your automated testing.\n")

    platform_to_test = get_valid_input(
        "What is the platform of the application under test? (Enter one of the following responses: Web Application, Mobile Application, API Services): ",
        ["Web Application", "Mobile Application", "API Services"])
```

2. The responses to the user's questions are taken and displayed for the user to see what they entered.

```
def get_user_input():
```

```
    user_responses = {
        "Platform to Test": platform_to_test,
        "Testing Requirements": testing_requirements,
        "Ease of Use": ease_of_use,
        # ... other criteria
    }
    return user_responses
```

3. The code implementation includes robust input validation that ensures that the given response options are entered by the user.

```
def get_valid_input(prompt, valid_options):
    while True:
        response = input(prompt)
        if response in valid_options:
            return response
        else:
            print(f"Invalid input! Please choose one of the following: {', '.join(valid_options)}")
```

4. The first stage after the user enters their responses is the tools filtering. As per the design of the application, the implementation includes a two-stage filtering process; the type of platform being tested and the type of test being done. Below shows how this is implemented in the code.

```
def filter_tools(tools, platform_to_test, testing_requirements):
    filtered_tools = []
    for tool in tools:
        if any(platform in tool["Platform"] for platform in platform_to_test) and \
                any(requirement in tool["Testing Requirement"] for requirement in testing_requirements):
            filtered_tools.append(tool)
    return filtered_tools
```

5. After the initial stage of tools filtering, the next set of questions address the criteria-based questions. The weights of each response linked to the different criteria are organised in a nested dictionary structure that maps qualitative inputs to numerical values. This is the structure of the criteria weights in the code implementation. An example in the sample code shows "Ease of Use" as a criterion and the possible responses of the users that will be converted from the qualitative response to the numerical value used for the WSM implementation.

```
criteria_weights = {
    "Ease of Use": {
        "Very Important": 1.0,
        "Important": 0.8,
        "Neutral": 0.6,
        "Not very important": 0.4,
        "Not Important": 0.2,
    },
    # ... other criteria
```

6. For each response, the user's input is received, and the weight is assigned in the code as per the below details.

```
def get_user_weights(user_responses, criteria_weights):
    user_weights = {}
    for criterion, response in user_responses.items():
        if criterion in criteria_weights:
            user_weights[criterion] =
criteria_weights[criterion][response]
    return user_weights
```

7. To ensure consistency, the weights are normalised in the code implementation as per the below details.

```
def normalize_weights(weights):
    total = sum(weights.values())
    if total > 0:
```

```
        return {k: v / total for k, v in weights.items()}
    return weights
```

8. The next stage after the weight assignment is the evaluation of the tools. The tools database structure is implemented using dictionaries and lists to represent the testing tools and their features. Below is the sample code representation. One of the tools, Appium, is in the sample code with the type of platforms for testing and the type of test Appium is used for.

```
tools = [
    {
        "Tool": "Appium",
        "Platform": ["Web Application", "Mobile Application"],
        "Testing Requirement": ["Functional", "Performance",
"Integration"]
    },
```

9. The various performance ratings of the automation testing tools for this study as indicated in *Table 4* are stored in a way that each tool maps to a list of ratings corresponding to the seven criteria. Below is a portion of how the code is implemented.

```
ratings = {
    "Appium": [0.9, 0.8, 0.8, 0.7, 0.6, 0.9, 0.8],
    "Selenium": [0.9, 0.9, 0.9, 0.8, 0.6, 0.8, 0.8],
    # ... other tools
}
```

10. The next step in the process is the score calculation. This happens after the tools are filtered based on the user's responses and the set of criteria questions are also answered.

```
def calculate_tool_scores(filtered_tool_names, user_weights,
ratings):
    tool_scores = {}
    for tool in filtered_tool_names:
        tool_scores[tool] = sum([
            user_weights.get("Ease of Use", 0) * ratings[tool][0],
```

```
            user_weights.get("Programming      Skills",       0)       *
ratings[tool][1],
            user_weights.get("Scalability    &    Performance",    0)    *
ratings[tool][2],
            user_weights.get("Flexibility    &    Customization",    0)    *
ratings[tool][3],
            user_weights.get("Licensing     and     Cost",      0)      *
ratings[tool][4],
            user_weights.get("Vendor Stability and Product Support", 0)
* ratings[tool][5],
            user_weights.get("Reviews    &    Recommendations",    0)    *
ratings[tool][6],
        ])
    return tool_scores
```

11. After the calculation is the result generation. Here the top three tools are displayed based on the tool with the highest scores per the platform and testing type and responses of the user.

```
def get_top_tools(tool_scores, top_n=3):
    sorted_tools = sorted(tool_scores.items(),
                          key=lambda x: x[1],
                          reverse=True)
    return sorted_tools[:top_n]
```

4.3.2 Web Application Version Implementation

The second implementation links an HTML and JavaScript code to an already existing AI model (Mistral AI) that recommends the best automated testing tool based on the user's response. The underlying selecting process was not a direct use of the WSM method as implemented in the Python code, however, the same concept was used. This method complements the Python code implementation because the number of testing tools available using the AI model is unlimited compared to the Python code implementation that had 14 tools.

For the web version of our design implementation, the system takes input from users based on specific questions asked via the user interface as per *Figure 11*. From the preferences of the users, weights are assigned to be used to compute the best score for the recommendation of the best automated testing tool. The implementation of the code starts with the user input collection. The same logic used for taking the user responses in the Python program is applied here in the web application. The main difference lies in the application of the WSM. In the web version, an AI prompt is used together with the user's responses to recommend the best automated tool using the WSM method which is displayed via the user interface. Appendix D has the detailed HTML code and the Java Script Code.

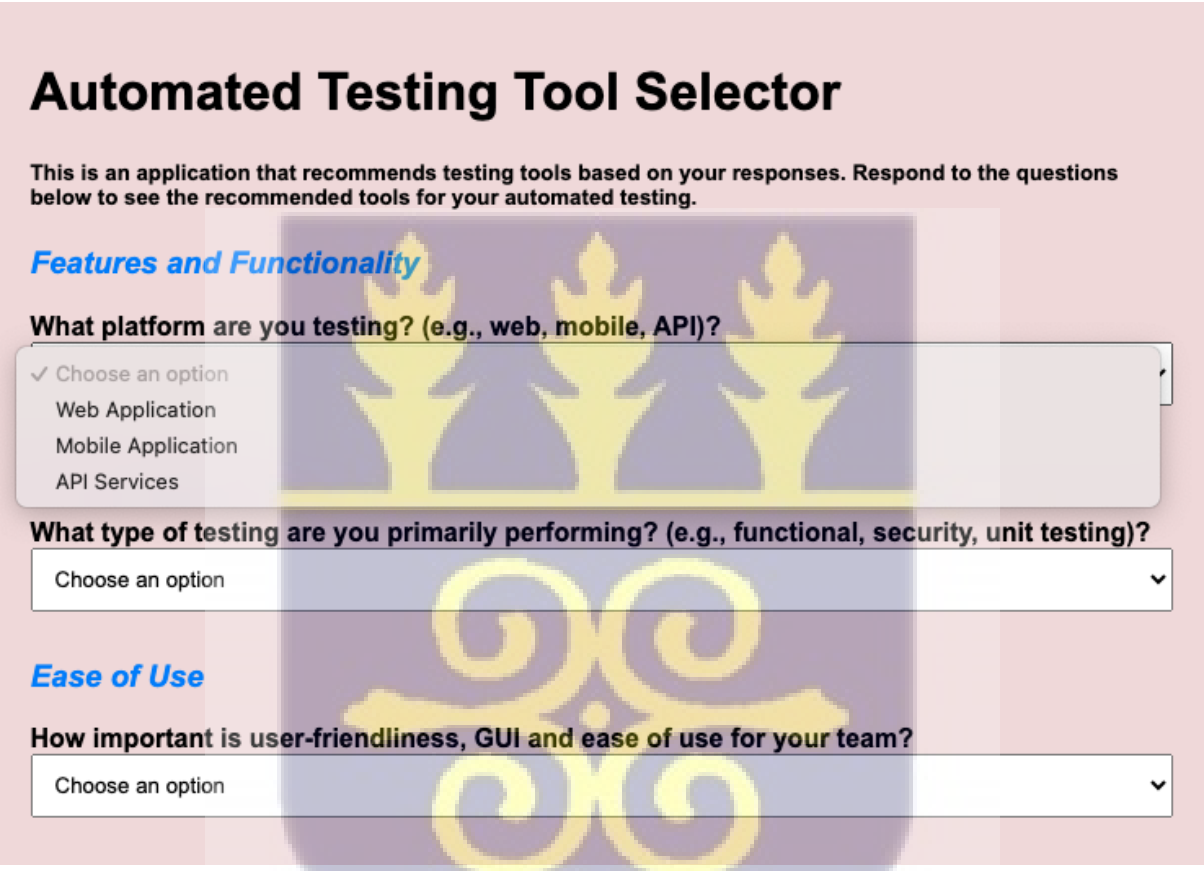

Figure 11: Web Version User Interface

4.4 System Testing and Results

The practical demonstrations of the developed applications were conducted across a number of scenarios. Based on the two main implementation approaches for this study, assessments were carried out using both the Python application and the AI-linked web version. For the Python

application, the 14 listed automated testing tools (alternatives) were applied across the defined criteria to validate the framework's effectiveness.

On the other hand, because the web application was linked to an already existing Mistral AI model, two main scenarios were initially done. One was done with the 14 alternatives of tools used for the Python code implementation. The other test was removing the limit of using only 14 testing tools to connect with available tools online. In order to do a better comparative analysis, the same user responses were used across the various scenarios. The platform to tests were changed to see the effects in the results.

4.4.1 Test Scenario 1: Testing Automation Tool Selector with Python Application I

With the first scenario of tests done, the Python application was used. The tool database was made up of the 14 tools selected for this study. The parameters used for the tests performed for this first test scenario are outlined in *Tables 7, 8 and 9*. The platform to test was changed for the three different tests done in this scenario. Results of the tests are indicated beneath each table.

Table 7: Python Application Test 1 Parameters for Mobile App Testing

Platform to test	Mobile Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important

Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Appium 2. Espresso 3. XCUITest

Table 8: Python Application Test 2 Parameters for Web App Testing

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Selenium 2. Cypress 3. JMeter

Table 9: Python Application Test 3 Parameters for API Testing

Platform to test	API Services
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner

Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Postman 2. SoapUI 3. JMeter

4.4.2 Test Scenario 2: Testing with AI-linked Web Version I

For the first set of tests with the web-version linked to the Mistral AI model, the prompt was used to mimic the Python application implementation, using the same number of testing tools, 14. *Tables 10, 11 and 12* outline the parameters used for the tests performed for this test scenario. Again, for the series of tests done, the platform to test was changed for the three different tests done in this scenario. Results of the tests are indicated beneath each table.

Table 10: Web Version Test 1 Parameters for Pre-defined Tools

Platform to test	Mobile Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important

Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Appium 2. Espresso 3. XCUITest

Table 11: Web Version Test 2 Parameters for Pre-defined Tools

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Selenium 2. Cypress 3. Puppeteer

Table 12: Web Version Test 3 Parameters for Pre-defined Tools

Platform to test	API Services
Testing Requirement	Functional Testing
Criteria	**User's Responses**

Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Postman 2. SoapUI 3. Rest Assured

4.4.3 Test Scenario 3: Testing with AI-linked Web Version II

For the next test scenario, the web version linked to the Mistral AI model was used to test but this time, the selected 14 tools used for the python application were not used. The prompt used allowed for the selection of tools to be opened to available testing tools on the market apart from the 14 chosen for the Python implementation. This was done to remove the limit of having to select from only 14 available tools.

Table 13: Web Version Test 1 Parameters without Pre-defined Tools

Platform to test	Mobile Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes

Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Appium 2. Calabash 3. Robot Framework

Table 14: Web Version Test 2 Parameters without Pre-defined Tools

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Selenium WebDriver 2. Cypress 3. Test Cafe

Table 15: Web Version Test 3 Parameters without Pre-defined Tools

Platform to test	API Services
Testing Requirement	Functional Testing

Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Rest Assured 2. JMeter 3. Postman

4.4.4 Test Scenario 4: Testing with AI-linked Web Application III

The last test scenario with the web version linked to the Mistral AI tests was done by directly asking Mistral AI to recommend automated testing tools. This was to see how the WSM-based tool developed compared with an already existing natural language model or AI model. The main prompt used was "Recommend the best three mobile (web or API) functional testing tools." For these tests, there was no information on the user preferences which is converted to weights unlike the initial tests done.

Table 16: Web Version Test 1 Parameters for General AI Prompt

Platform to test	Mobile Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important

Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Appium 2. Espresso 3. XCTest

Table 17: Web Version Test 2 Parameters for General AI Prompt

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Selenium WebDriver 2. Cypress 3. Playwright

Table 18: Web Version Test 3 Parameters for General AI Prompt

Platform to test	API Services
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended): 1. Postman 2. Karate 3. Rest Assured

4.4.5 Test 5: Sensitivity Tests on the WSM-based Python Application

To affirm the choice of MCDM method used, sensitivity tests were done. Different user responses for the same time of test and platform to test were explored. This was to check the sensitivity of the WSM using the system built. This test scenario was also done to see how the changes in user responses affect the weights and subsequently the ranking of the tools recommended. Five tests were done. The parameters used for the tests are indicated in *Tables 19, 20, 21, 22 and 23.* The responses that were changed have been highlighted in yellow. Each test was a build-up of the previous test conducted.

Table 19: Sensitivity Test 1 Parameters for Python Application

Platform to test	Web Application
Testing Requirement	Functional Testing

Criteria	**User's Responses**
Ease of Use	Not Important
Programming Skills	Intermediate
Scalability and Performance	Yes
Flexibility and Customization	No
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Very Important
Reviews And Recommendations	Very Important

Results (Tools Recommended and their scores):

Tool: Selenium, Score: 0.800

Tool: JMeter, Score: 0.791

Tool: TestComplete, Score: 0.782

Table 20: Sensitivity Test 2 Parameters for Python Application

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Not Important
Programming Skills	Advanced
Scalability and Performance	No

Flexibility and Customization	No
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Neutral
Reviews And Recommendations	Very Important

Results (Tools Recommended and their scores):

Tool: JMeter, Score: 0.790

Tool: Selenium, Score: 0.785

Tool: TestComplete, Score: 0.770

Table 21: Sensitivity Test 3 Parameters for Python Application

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Open Source (Free)
Vendor Stability and Product Support	Neutral
Reviews And Recommendations	Very Important

Results (Tools Recommended and their scores):

Tool: Selenium, Score: 0.817

Tool: JMeter, Score: 0.800

Tool: TestComplete, Score: 0.799

Table 22: Sensitivity Test 4 Parameters for Python Application

Platform to test	Web Application
Testing Requirement	Functional Testing
Criteria	**User's Responses**
Ease of Use	Very Important
Programming Skills	Beginner
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Commercial
Vendor Stability and Product Support	Neutral
Reviews And Recommendations	Neutral

Results (Tools Recommended and their scores):

Tool: Selenium, Score: 0.834

Tool: TestComplete, Score: 0.815

Tool: JMeter, Score: 0.807

Table 23: Sensitivity Test 5 Parameters for Python Application

Platform to test	Web Application
Testing Requirement	Functional Testing

Criteria	User's Responses
Ease of Use	Very Important
Programming Skills	Advanced
Scalability and Performance	Yes
Flexibility and Customization	Yes
Cost and Licensing	Commercial
Vendor Stability and Product Support	Neutral
Reviews And Recommendations	Neutral

Results (Tools Recommended and their scores):

Tool: Selenium, Score: 0.829

Tool: TestComplete, Score: 0.816

Tool: JMeter, Score: 0.808

4.4.6 Test 6: Validation Tests with Industry Experts

After the sensitivity tests, 52 software testers, quality assurance engineers, and industry experts tested the model to validate the system built. Their tests were done using the web version linked to the Mistral AI model with an unlimited number of tools. *Table 24* contains a sample of the data collected. This shows the responses of five (5) different testers, their preferences, and the actual results against their expected results. The results that matched the expectations of the testers have been highlighted.

4.5 Analysis and Discussion of Results

The focus of this study was to use the WSM method to evaluate testing tools and recommend the best testing tool based on the user's response to criteria-based questions. In this section, the results of the various tests performed are discussed.

Table 24: Validation Test Sample Parameters and Results

Platform to Test	**Type of Test**	**Criteria Importance**	**Expected Results of Tool (s) to Recommend**	**Actual Results of Tools Recommended**
Mobile App	Performance	Ease of Use: Very Important Programming Skills: Beginner Scalability Needed: Yes Flexibility Needed: Yes License Type: Open Source Product Support: Important Reviews Given: Important	• Apache JMeter	• Locust • Apache JMeter • Gatling
Mobile App	Security	Ease of Use: Important Programming Skills: Advanced Scalability Needed: Yes Flexibility Needed: Maybe License Type: Open Source Product Support: Important Reviews Given: Important	• OWASP Zed Attack Proxy (ZAP)	• OWASP Zed Attack Proxy (ZAP) • Mobile Security Framework • Quick Android Review Kit
Web App	Integration	Ease of Use: Very Important Programming Skills: Beginner Scalability Needed: Yes Flexibility Needed: Yes License Type: Commercial Product Support: Important Reviews Given: Important	• Selenium • Katalon Studio	• Selenium • TestComplete • Katalon Studio
Web App	Unit	Ease of Use: Important Programming Skills: Advanced Scalability Needed: Yes Flexibility Needed: No License Type: Open Source Product Support: Important Reviews Given: Important	• Junit	• Junit • TestNG • Mockito
API	Functional	Ease of Use: Important Programming Skills: Advanced Scalability Needed: Yes Flexibility Needed: Maybe License Type: Open Source Product Support: Important Reviews Given: Important	• Postman • Rest Assured	• Postman • Karate DSL • Rest Assured

4.5.1 Python Application Evaluation

From the first test scenario done with the Python application, each of the tools recommended were based on the platform to test (Mobile, Web or API) and the type of test (functional) used. Checking from the tools database used, the filtering module worked as expected. From the tools filtered, it was seen that the tools recommended were based on the user responses to the criteria-based questions.

The first test done (parameters in *Table 7)* was on the mobile app functional testing tool recommendation. The results were Appium, Espresso and XCUITest. All three tools were in the database and are all used for mobile app testing. When it came to the web application functional testing tools recommendation (parameters in *Table 8*), the three tools in the results were Selenium, Cypress and JMeter. All three tools recommended from the tools database show that they are all tools for testing web applications.

A similar observation was made for the API services functional testing (parameter in *Table 9*). From these results, the design of this work played the key role of evaluating the filtered tools successfully according to the platform to test and the type of testing. After the tool filtering process, the WSM method was applied to recommend the best testing tools based on the user's responses and the filtered tools with the highest scores.

4.5.2 AI-Linked Web Version Results Discussion

For the web version of the system implementation, the first test scenario was done mimicking the Python code implementation. The results of all the similar tests done with the Python application and the web version is shown in *Table 25*. The results of the web version showed similar recommendations made by the Python application when it come to the mobile app functional testing. Both the Python application and the web version gave Appium, Espresso and XCUITest as the best tools for testing.

However, when it came to the web app functional test testing tool recommendation, two of the recommendations made were similar to the results of the Python code implementation. Thus, Selenium and Cypress were recommended by both. One of the tools was different. JMeter was recommended for the Python app, whereas the web version recommended Puppeteer. For the API platform functional test tool recommendation too, a similar observation was made. Apart from Postman and SoapUI being recommended for both implementations, JMeter differed from Rest Assured in the web version results.

The possible reason for this slight variation could be because the AI-linked web version is prompted to use the WSM method to recommend the best testing tools. In comparison to the Python code, the tools database for the web version varies because it is linked to the Mistral AI model. This is why the variations were expected. But in all, the WSM method used recommended tools that were all based on the type of test or platform to test which shows that the methodology is valid.

The next test scenario done for the web application was done evaluating the testing tools available on the market. There was no limit of 14 testing tools as used in the Python code implementation. The results of these evaluations produced a different set of recommended tools, even though the type of test and testing platform remained the same as in the previous evaluation. From these results, as shown in *Table 25,* Test Scenario 3, it is seen that when the user responses to the criteria-based questions are connected to the Mistral AI, the number of testing tools to evaluate for the selection process is countless. The reason is that the AI model already has features of well-known testing tools to use in the evaluation and recommendation process.

Table 25: Tests Scenarios 1-4 Results Comparison

	Recommended Tools for Functional Testing		
Evaluation Scenario	**Mobile App Testing**	**Web App Testing**	**API Testing**
Scenario 1: Python application with user responses weights assigned	1. Appium 2. Espresso 3. XCUITest	1. Selenium 2. Cypress 3. JMeter	1. SoapUI 2. JMeter 3. Postman
Scenario 2: Web Application with AI mimicking Python Implementation	1. Appium 2. Espresso 3. XCUITest	1. Selenium 2. Cypress 3. Puppeteer	1. Postman 2. SoapUI 3. Rest Assured
Scenario 3: Web App with AI using WSM in the query and unlimited set of tools	1. Appium 2. Calabash 3. Robot Framework	1. Selenium WebDriver 2. Cypress 3. Test Cafe	1. Rest Assured 2. JMeter 3. Postman
Scenario 4: General AI prompt asking for the best testing tools	1. Appium 2. Espresso 3. XCTest	1. Selenium WebDriver 2. Cypress 3. Playwright	1. Postman 2. Karate 3. Rest Assured

Considering that the AI model played a role in enhancing the system built, the last test was done to see if there was a need to connect the user responses, which aids in the weight's assignment. From this test, it was observed that it was easy for the AI to suggest the best testing tools. From the results in *Table 25,* Test Scenario 4, the responses were similar to the responses received during the AI-linked web version tests.

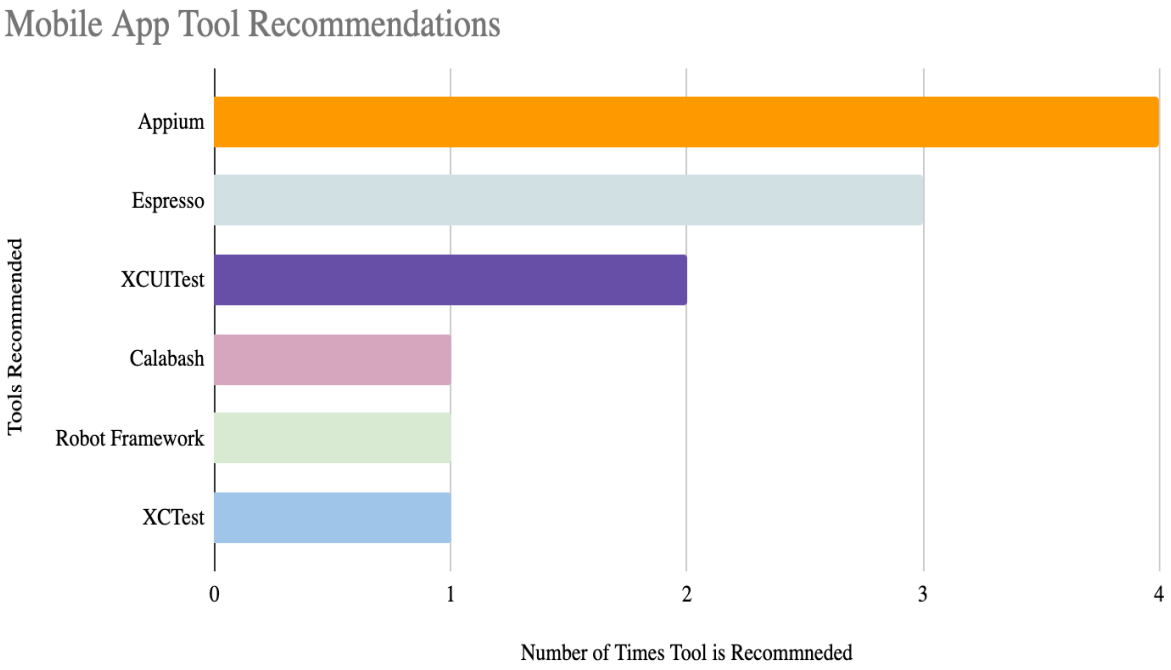

Figure 12: Mobile App Test Tool Recommendation Results

Figure 13: Web App Test Tool Recommendations Results

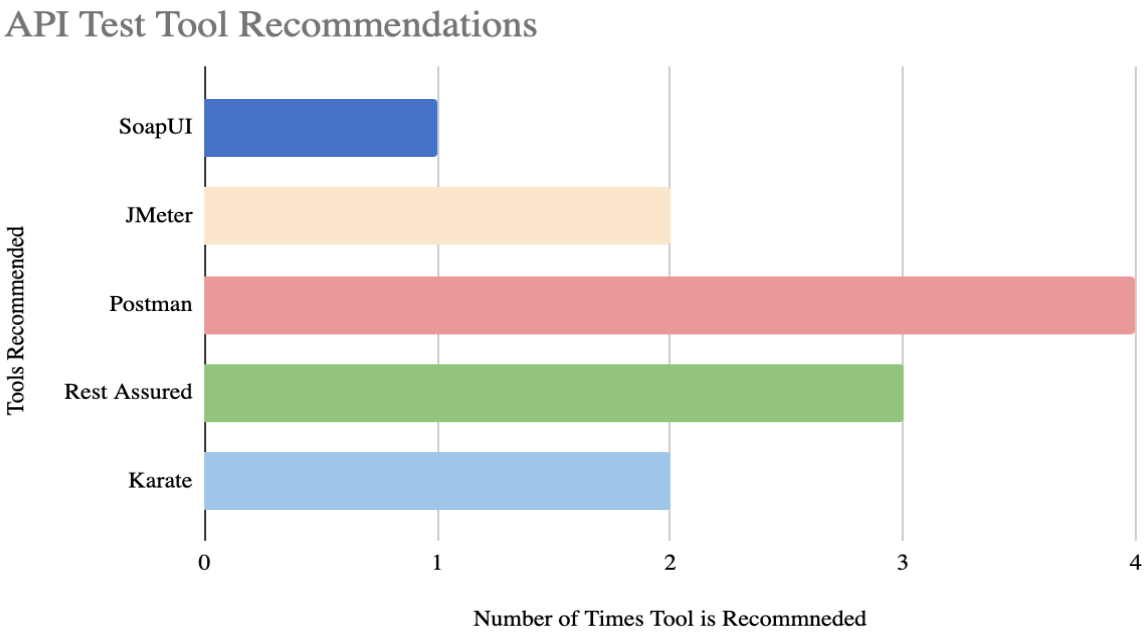

Figure 14: API Test Tool Recommendation Results

From the results in Table 25, the AI model alone could not give customised recommendations for specific requirements that were considered in the questions, especially when the user responses were tweaked to accommodate specific criteria levels of importance. *Figures 12, 13, and 14* show the compared results of the tool recommendations for these four scenarios have been put in

4.5.3 Sensitivity Tests Results Analysis

Apart from the web testing and the Python code implementation tests, sensitivity tests were done to check how the WSM method used for this study does when user responses are changed. This evaluation was necessary because of the weight-based user responses which determined the tools recommended. To validate the assertion of the approach used, some of the user responses were changed one at a time. *Tables 19, 20, 21, 22 and 23* contains the parameters used for the tests done to check the sensitivity of the model used.

The sensitivity tests results are indicated in *Table 26.* From the results, Selenium, JMeter and TestComplete were the tools recommended. It is seen that these 3 tools remained the same throughout the tests results. This is because the type of test and platform to test were not changed for all the five tests done. However, even though the tests were done with the same

platform to test and the same type of test, thus, web application functional testing, the scores of each recommended tool kept changing for all the five tests done.

For the first sensitivity test done, Selenium ranked first with a score of 0.8 whereas the second test results had JMeter coming first with a score of 0.79. For each user response adjustment, the scores changed, and the ranking of the tools changed to a different order. This proves that tools recommended are based on the weights assigned per the user's response. Figure 15 shows the analysis of the sensitivity test results.

Table 26: Sensitivity Test Results

Simulation	Rank	Tool	WSM Score
1	1st	Selenium	0.8
1	2nd	JMeter	0.791
1	3rd	TestComplete	0.782
2	1st	JMeter	0.79
2	2nd	Selenium	0.785
2	3rd	TestComplete	0.77
3	1st	Selenium	0.817
3	2nd	JMeter	0.8
3	3rd	TestComplete	0.799
4	1st	Selenium	0.834
4	2nd	TestComplete	0.815
4	3rd	JMeter	0.807
5	1st	Selenium	0.829
5	2nd	TestComplete	0.816
5	3rd	JMeter	0.808

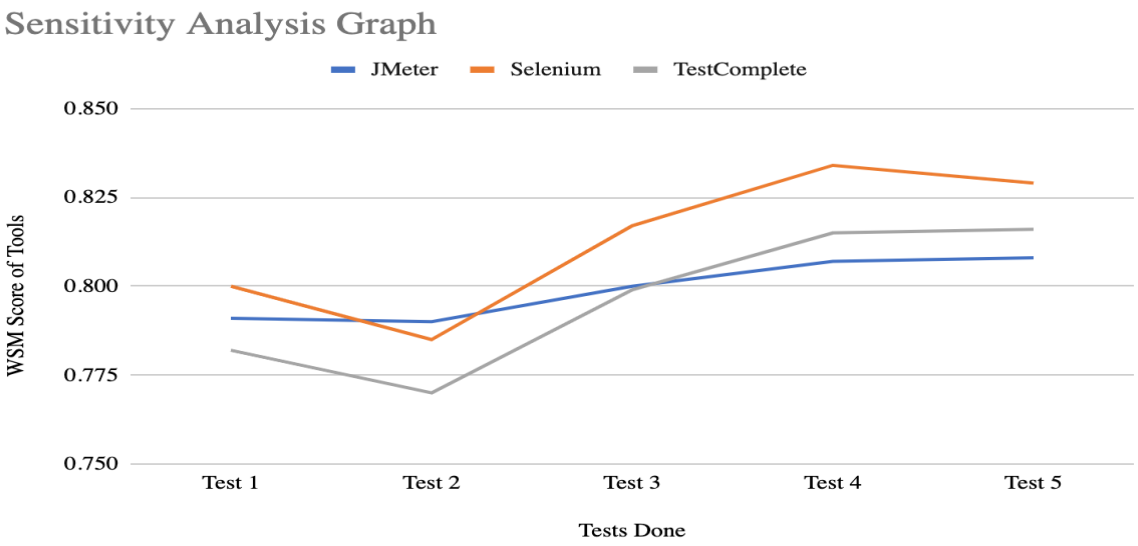

Figure 15: Sensitivity Tests Results Analysis

4.5.4 Validation Test Results Analysis

Validation tests were conducted by 52 industry experts. Over 50% of the participants were from the tech industry, about 30% from FinTech and the remaining from banking and other industries. The background of these experts is displayed in *Figure 16*.

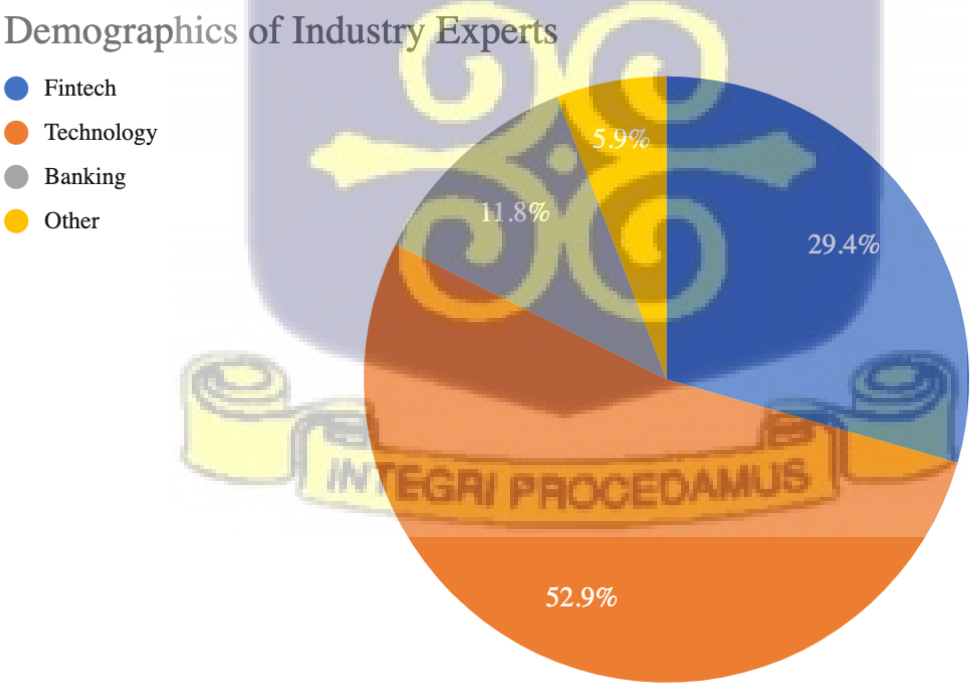

Figure 16: Demographics of Industry Experts who tested the System

Based on the tests done by the industry experts, questions as indicated in Appendix A, were asked to get the feedback of the testers. Questions about the efficiency, scalability, accuracy of the system, among others were asked. The feedback received from the questions after the system was tested is represented in *Figure 17.*

From these results, it is seen that over 50% of the testers saw the system to be useful, scalable, efficient and accurate. Over 80% indicated that the tool was relevant whereas 86% of the testers mention, the system was easy to use. When it comes to the scalability though, 46% of the testers agreed that the system as scalable which means that improving the system to make it scalable is key. All in all, the systematic way of making the decision of automation tool selection is seen to be important and the use of the MCDM method makes it more efficient. From these results and feedback, there is an assurance that with little improvements as indicated in the recommendation of this study, the system can be optimized.

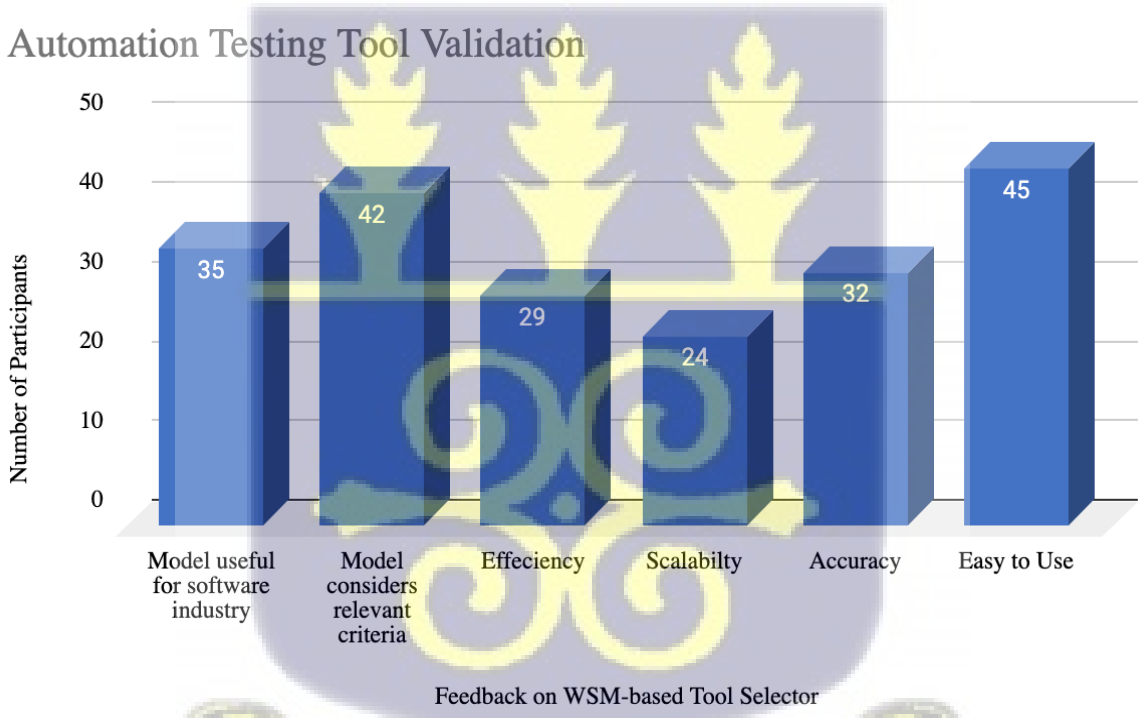

Figure 17: Feedback on Tool from Industry Experts

4.5.5 Data Analysis on User Responses During Testing

Data analysis was done on the responses used to run the WSM-based application during the validation tests. *Figure 18* is a representation of the responses of the testers. This was in response to the user preference on platform to tests. It is seen that 24 out of the 52 respondents chose mobile app testing as their platform to test. 19 chose web app as the platform to test and

9 chose API as the platform to test. This outcome is not surprising considering that a lot of applications of late are accessed on mobile. From these results, it can be said that most testers place focus on mobile and web testing compared to API testing.

The type of test choses is represented in *Figure 19.* Functional testing was counted as the highest followed by security and unit testing. This information affirms the choice made on the 5 types of software testing selected for this study. It is seen that performance testing had the least count. It can be deduced that there is much focus on functional, security and unit testing.

Figure 20 shows further analysis done on the platform to test and the type of testing. For mobile app tests, it is seen that focus is placed on functional and security testing. For mobile app testers, based on the respondents of this study, performance testing is minimal.

Figure 21 shows the analysis of the types of tests done on web application platforms. Functional tests had the highest count followed by performance testing. This shows that for web applications, performance testing plays a major role.

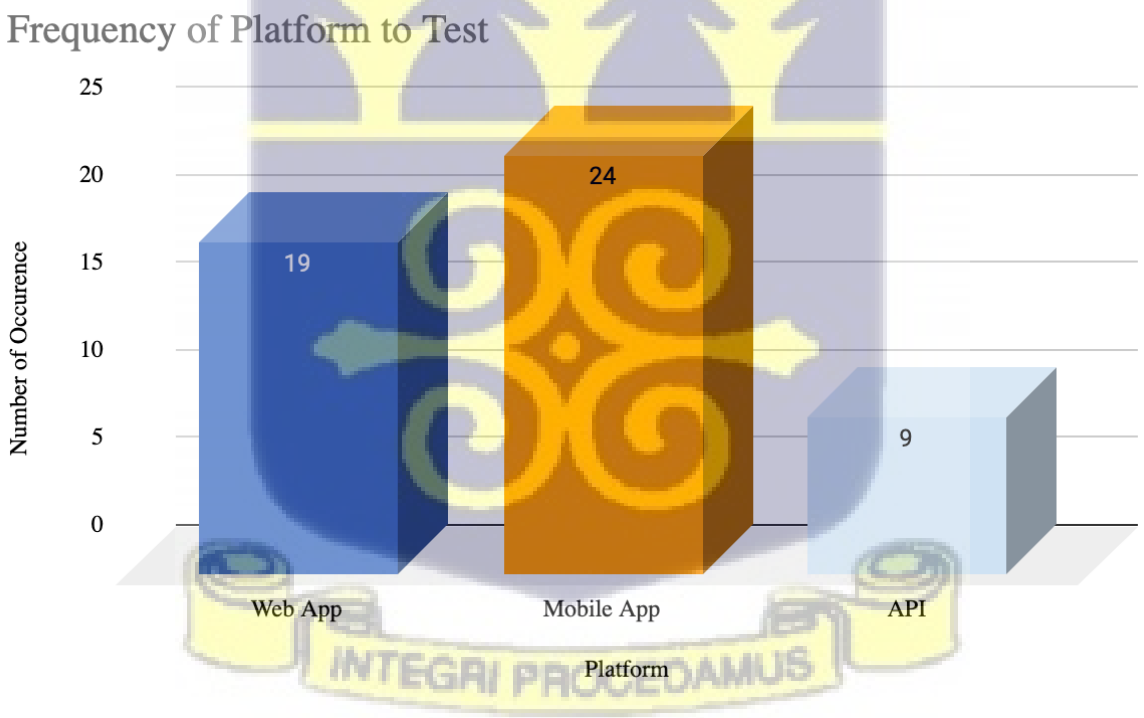

Figure 18: Analysis on Platform to Test

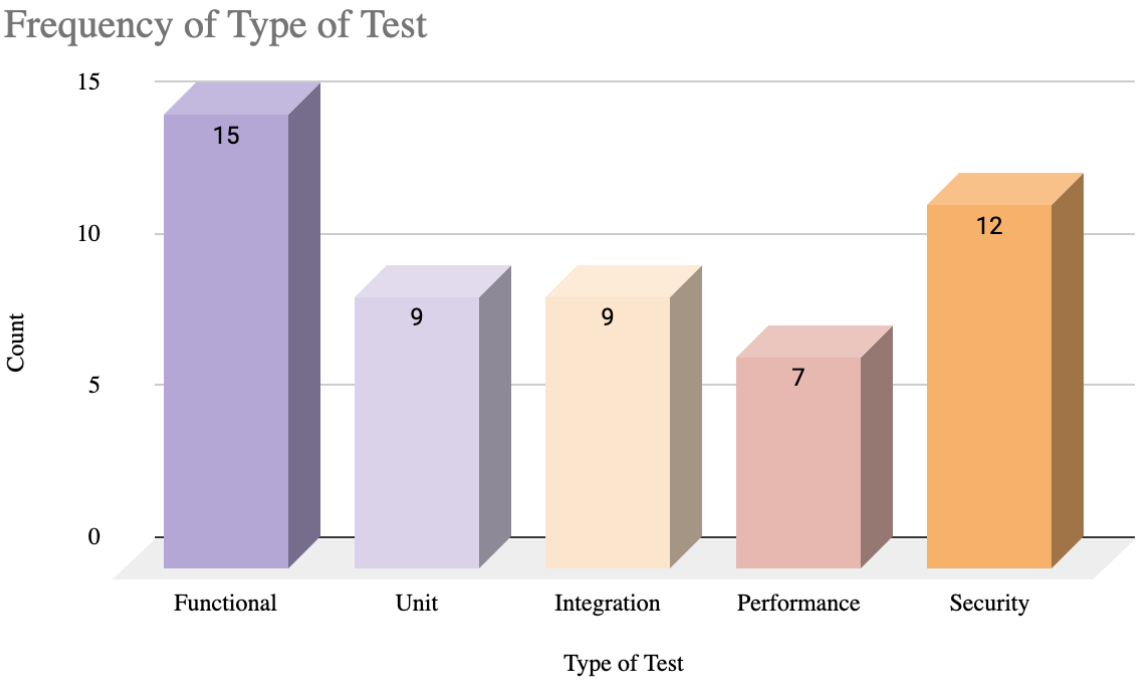

Figure 19: Analysis on Type of Test

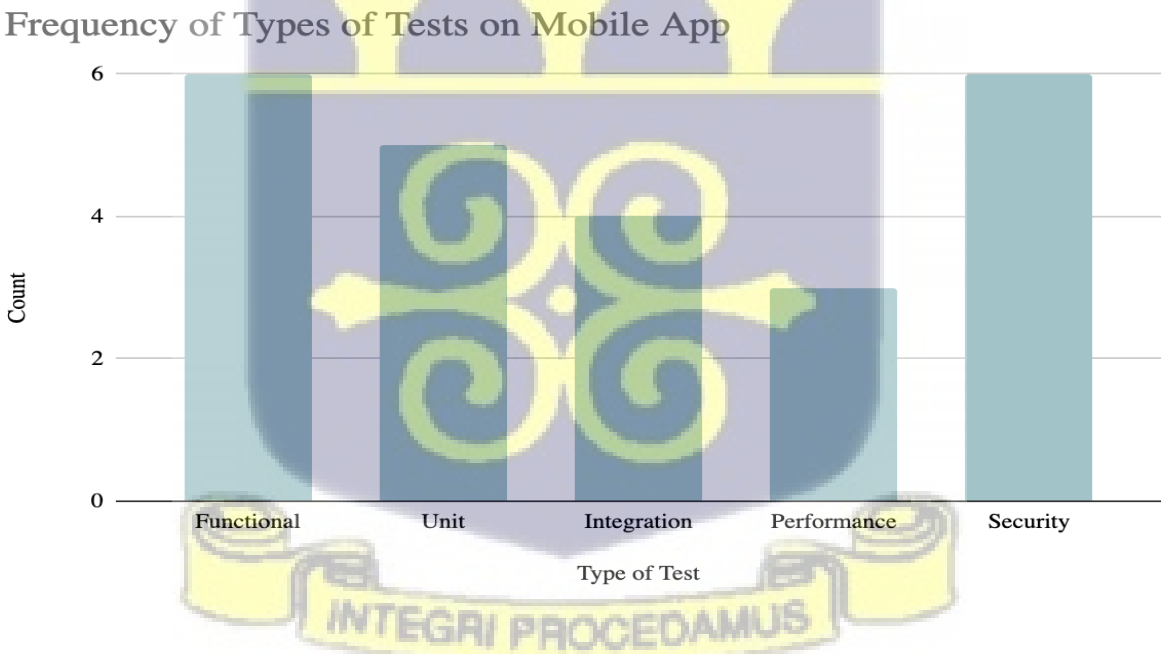

Figure 20: Analysis of Type of Tests done on Mobile App

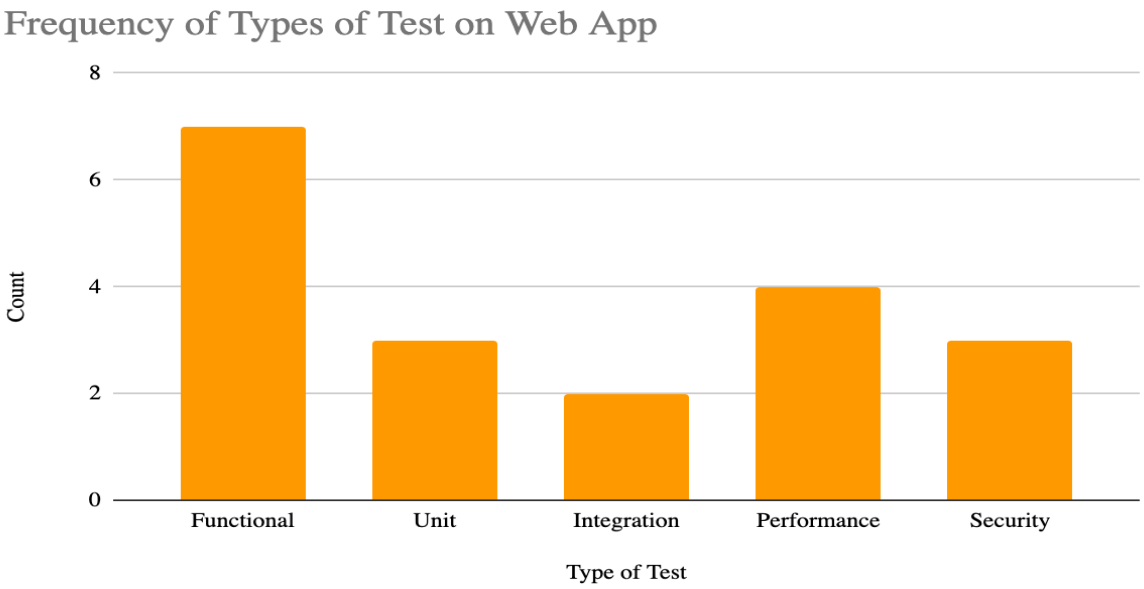

Figure 21: Analysis on Type of Tests done on Web App

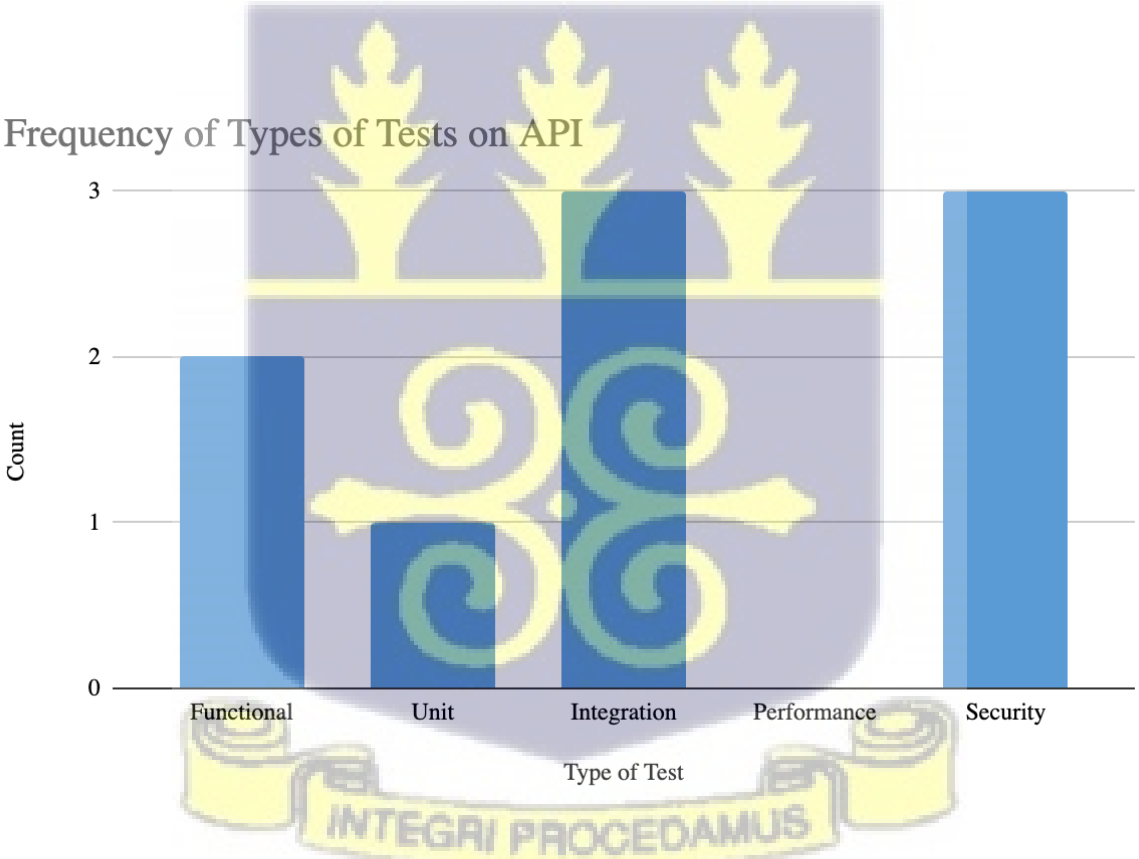

Figure 22: Analysis on Type of Tests done on API

Thirdly, *Figure 22* shows the analysis of the data collected for those who chose API as the platform to test. It is seen that attention is given to API security and integration testing. This can be attributed to the fact that, with the advancement of technology, many tech products

integrations are done via APIs hence the focus on integration and security tests for APIs. Performance testing on the other hand did not have any counts which affirms the focus on web app performance testing unlike the API performance testing.

Figure 23 is analysis on the first criteria question. This was about the ease of use of the test automation tool. Most of the respondents indicated that it was important or very important to consider ease of use for selecting automated testing tool.

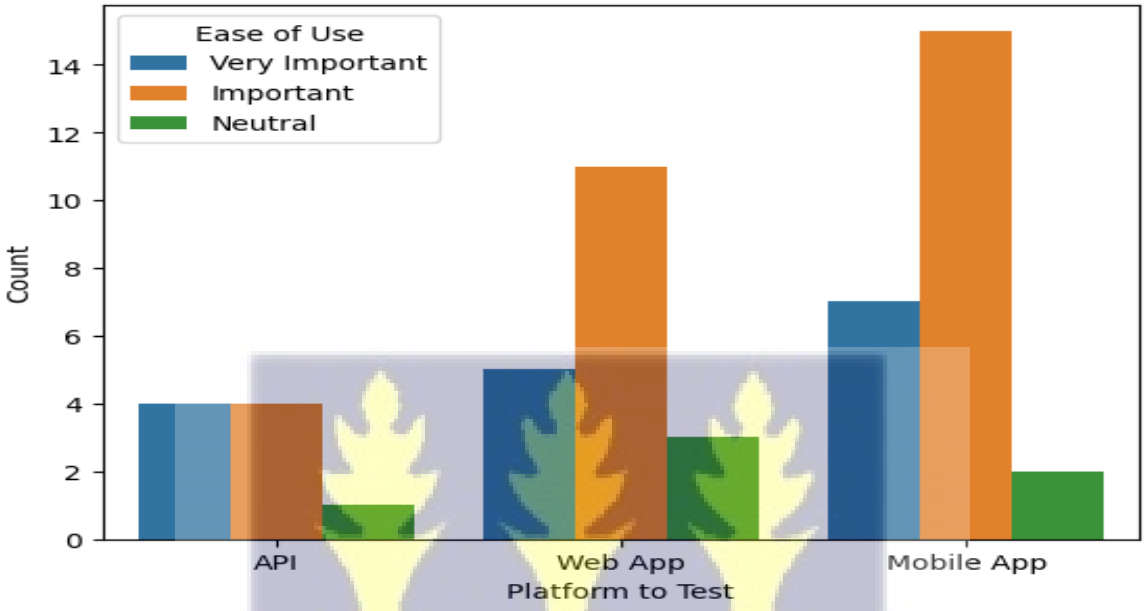

Figure 23: Criteria Analysis - Ease of Use

Figure 24 gives analysis on the second selected criteria for this study, programming skills. For those who indicated API as the platform to test, skills at beginner level were of importance compared to advanced to intermediate levels. For those who selected web app as the platform to test, an equal count was made on the level of beginner and advanced level of skills. For mobile app platform however, the focus of testers is on people with advanced lever of programming skills. This affirms the trends seen earlier on how focus is placed on mobile testing.

Figure 25 shows the analysis on the next criteria, scalability and performance of testing tool. Over 70 percent of the respondents of each platform indicated that they would want a tool that is scalable and performs well. This supports the choice of selecting this criterion for the study.

In *Figure 26,* the analysis on the next criteria which is flexibility and customization of automated testing tools is given. From the responses, it is seen that not a lot of emphasis is placed on the customization or flexibility of the automated testing tool during the selection

process. Most of the responses indicated that if the tool has that ability, they will be fine with it based on the response of "Maybe" having the highest count for all the three test platforms.

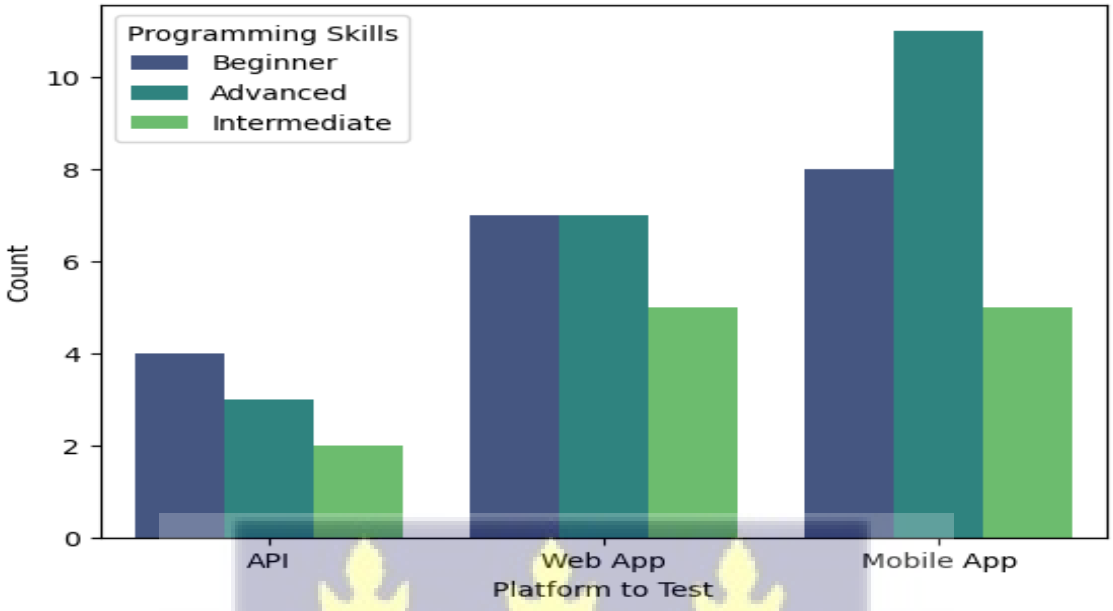

Figure 24: Criteria Analysis – Programming Skills

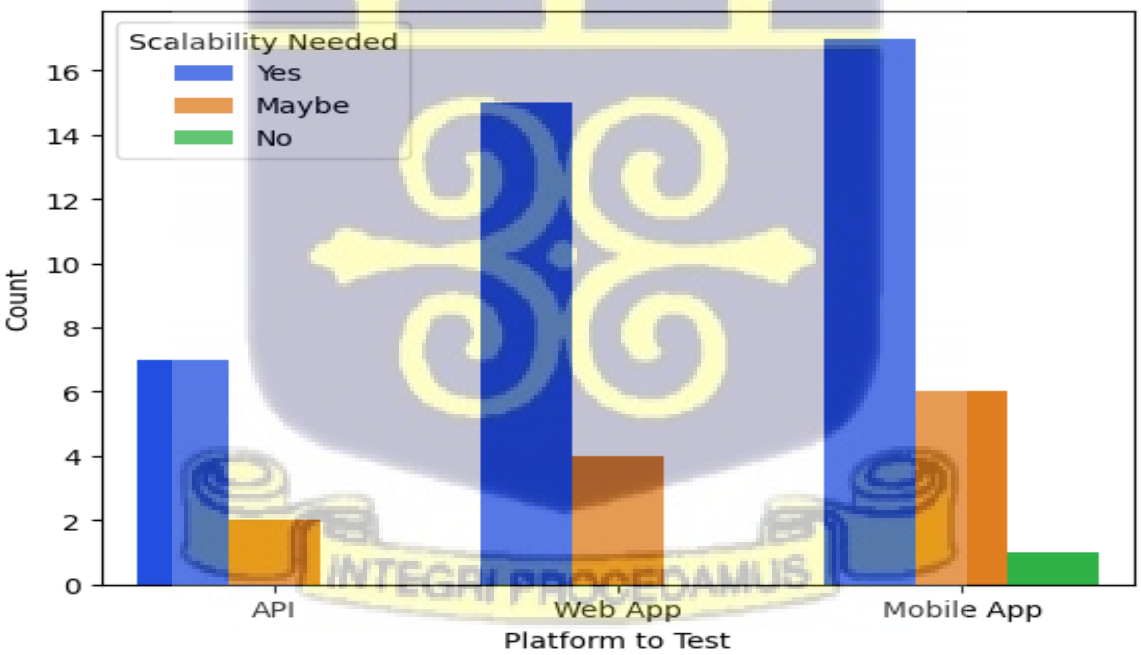

Figure 25: Criteria Analysis - Scalability and Performance

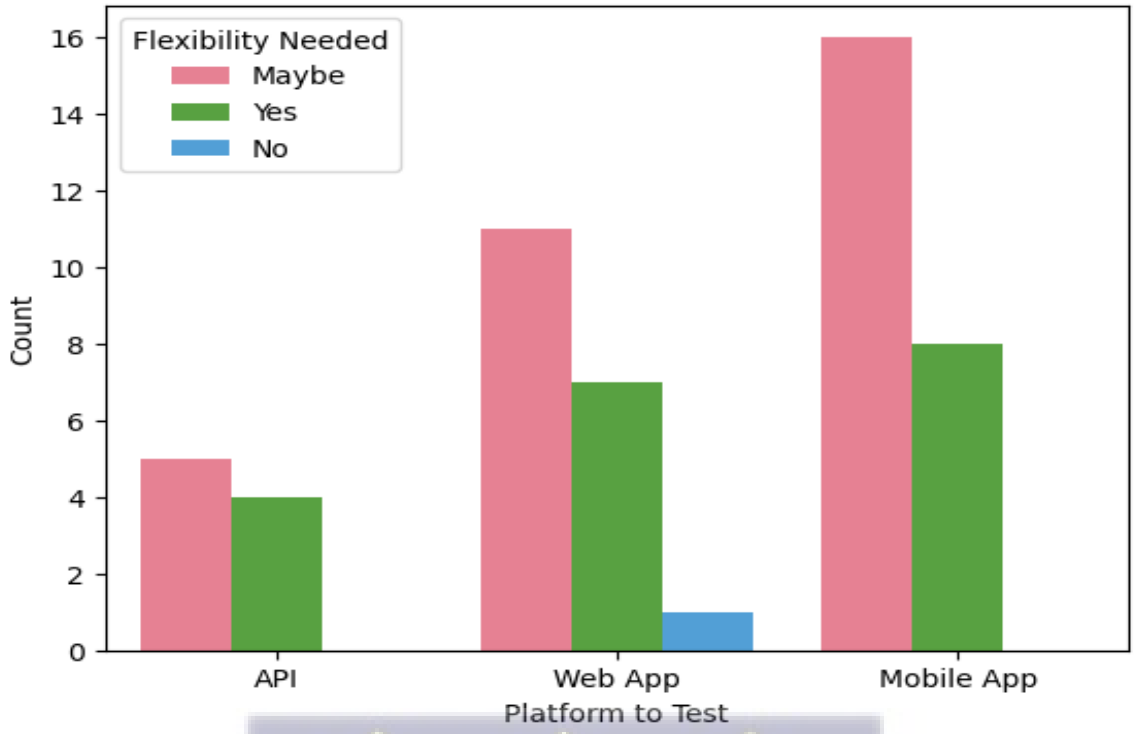

Figure 26: Criteria Analysis - Flexibility and Customization

The next criteria analysis is on the licensing and cost as shown in *Figure 27.* It was observed that most testers are interested in using open-source applications. This is no surprise as most open-source tools are available for free and, there is community support given to their users.

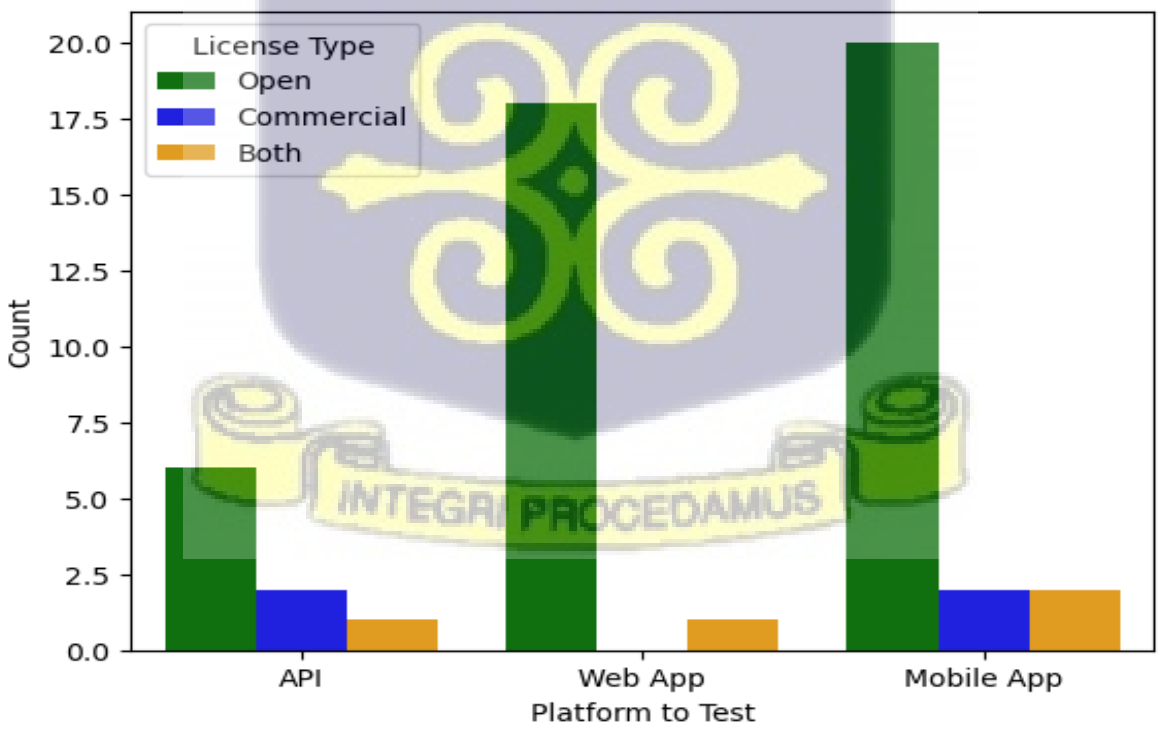

Figure 27: Criteria Analysis - License and Cost

The next criteria analysis on the product support and vendor stability is shown in *Figure 28.* Majority of the testers were keen on product support and vendor stability. With the use of any application, everyone wants to have access to the application when they need to use it. This is why more than 80% of the respondents indicated vendor stability and the product support as an important or very important for their tool selection.

Figure 29 gives the final analysis on the criteria, reviews and recommendations. The data shows a mix of responses based on the three platform to tests. For all the three platforms to test, this criterion was of importance. However, it is seen that 10 of the respondents do not see this criterion as an important factor for making a choice of the testing tool to use. This can be attributed to the fact that some tools may not have reviews or recommendations but then can work well for a particular testing type.

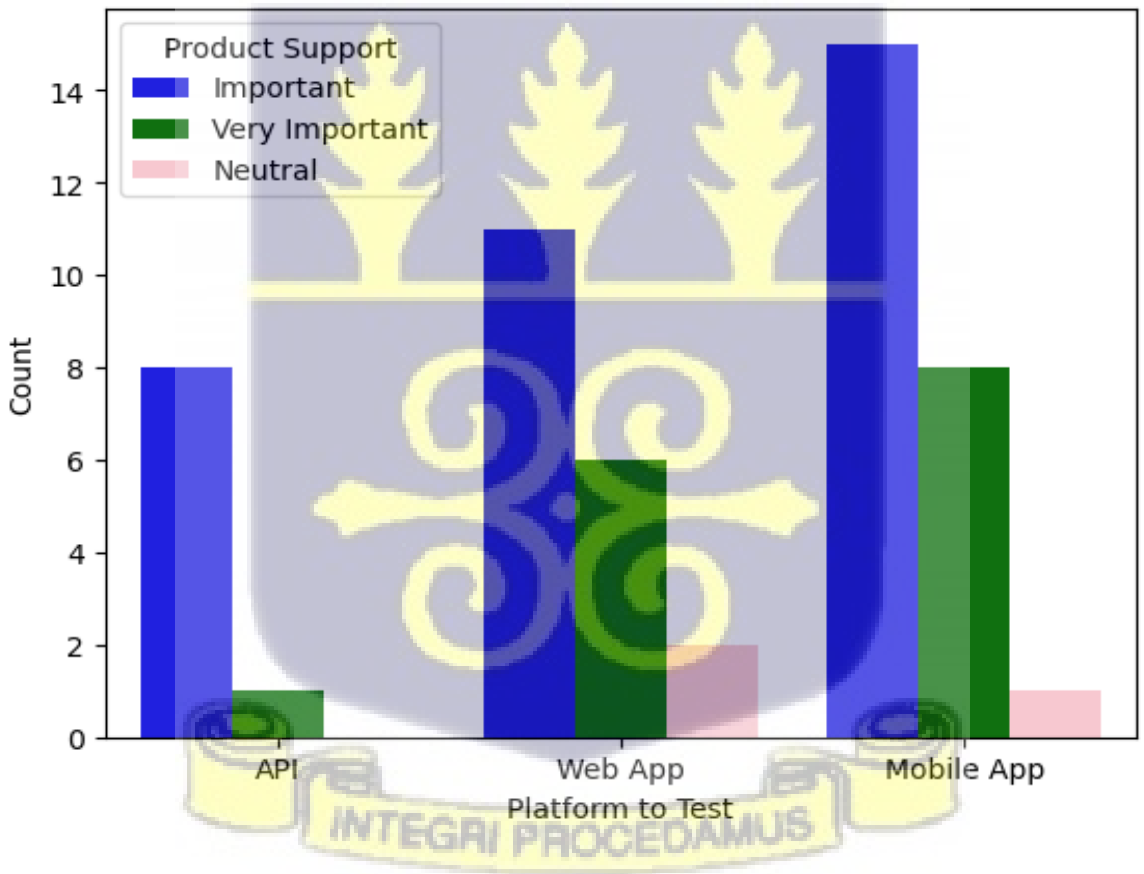

Figure 28: Criteria Analysis - Product Support and Vendor Stability

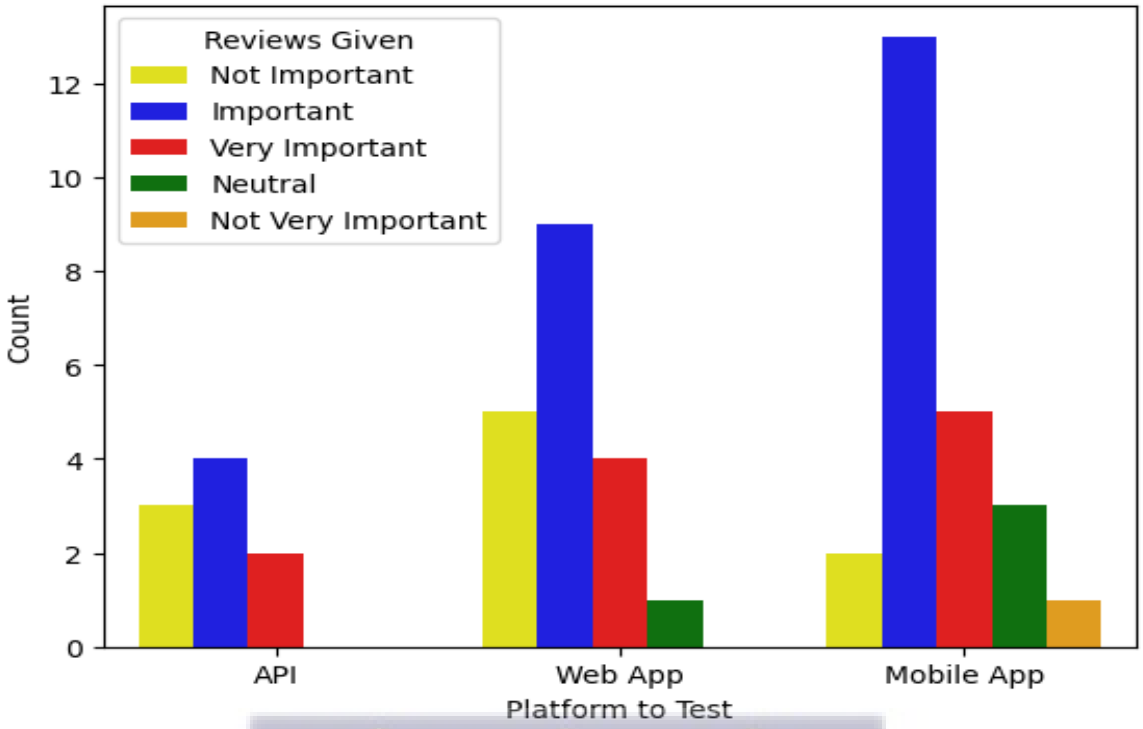

Figure 29: Criteria Analysis - Reviews and Recommendations

4.6 Comparative Analysis and Evaluation

One key thing crucial for this study was to compare existing methods of selecting automated testing tools with the developed application and approach discussed in this study. The next section compares the WSM approach to selecting automated testing tools and two existing methods.

4.6.1 AHP MCDM Method Comparison

One major comparison that was necessary was the use of AHP method, another MCDM method for testing tool selection. For the same set of tools and criteria used in this study, AHP method was used to determine the right automated testing tools and did some comparisons. For the AHP method, weights had to be assigned the various criteria by using a pairwise comparison. This method in the application of the AHP method limited the dynamic weight assignments per user's preference which is the innovation of this study.

Apart from the sensitivity tests performed on the WSM method, the AHP method had to be tested to compare the two different methods and the results as pertaining to the aim of this research. *Table 27* is the list of seven criteria and their respective weights from the pairwise

comparison of criteria. The consistency index was checked for these outcomes and confirmed to be acceptable. From the weights, tests were run to see the effect of user preferences on the AHP score.

Table 27: Criteria and Weights for AHP Method

Criteria	**Weight**
Cost and Licensing	0.043
Programming Skills	0.114
Flexibility and Customization	0.101
Ease of Use	0.260
Scalability and Performance	0.232
Vendor Stability and Product Support	0.143

4.6.1.1 AHP Tests

These tests were done to check how the AHP method compares with the proposed WSM method for this goal of this research.

Table 28AHP Test 1 with Pre-defined Weights

Platform to test	Mobile Application	
Testing Requirement	Functional Testing	
Criteria	**User's Responses**	**Criteria Weight**
Ease of Use	Very Important	0.260
Programming Skills	Beginner	0.114
Scalability and Performance	Yes	0.232
Flexibility and Customization	Yes	0.101
Cost and Licensing	Open Source (Free)	0.043

Vendor Stability and Product Support	Very Important	0.143
Reviews And Recommendations	Very Important	0.108

Results (Tools Recommended and their scores):

Tool: Appium, Score: 0.0739

Tool: Espresso, Score: 0.0748

Tool: TestComplete, Score: 0.0668

Table 29: AHP Test 2 with Pre-defined Weights

Platform to test	Mobile Application	
Testing Requirement	Functional Testing	
Criteria	**User's Responses**	**Criteria Weight**
Ease of Use	Very Important	0.260
Programming Skills	Advanced	0.114
Scalability and Performance	No	0.232
Flexibility and Customization	No	0.101
Cost and Licensing	Commercial	0.043
Vendor Stability and Product Support	Not Very Important	0.143
Reviews And Recommendations	Important	0.108

Results (Tools Recommended and their scores):

Tool: Appium, Score: 0.0739

Tool: Espresso, Score: 0.0748

Tool: TestComplete, Score: 0.0668

Table 30: WSM Test 1 to Compare AHP Test

Platform to test	Mobile Application	
Testing Requirement	Functional Testing	
Criteria	**User's Responses**	**Criteria Weight**
Ease of Use	Very Important	0.143
Programming Skills	Beginner	0.143
Scalability and Performance	Yes	0.143
Flexibility and Customization	Yes	0.143
Cost and Licensing	Open Source (Free)	0.143
Vendor Stability and Product Support	Very Important	0.143
Reviews And Recommendations	Very Important	0.143

Results (Tools Recommended and their scores):

Tool: Appium, Score: 0.0737

Tool: Espresso, Score: 0.0722

Tool: TestComplete, Score: 0.0664

Table 31: WSM Test 2 to Compare AHP Test

Platform to test	Mobile Application	
Testing Requirement	Functional Testing	
Criteria	**User's Responses**	**Criteria Weight**
Ease of Use	Very Important	0.217

Programming Skills	Advanced	0.130
Scalability and Performance	No	0.130
Flexibility and Customization	No	0.130
Cost and Licensing	Commercial	0.130
Vendor Stability and Product Support	Not Very Important	0.087
Reviews And Recommendations	Important	0.174

Results (Tools Recommended and their scores):

Tool: Appium, Score: 0.0777

Tool: Espresso, Score: 0.0681

Tool: TestComplete, Score: 0.0708

4.6.1.2 AHP Tests Results Analysis

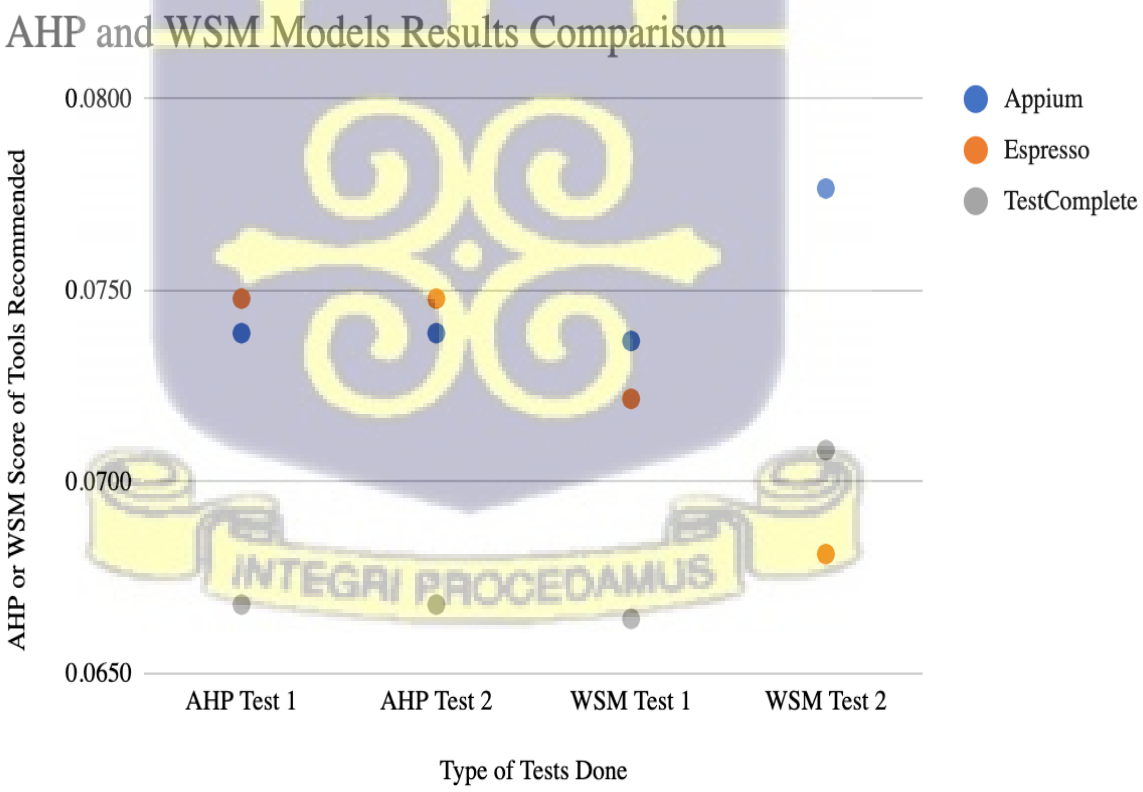

Figure 30: AHP and WSM Test Results Comparison

Figure 30 above is the results of tests done using both the AHP and WSM methods. The AHP method, although a very good MCDM method compared to the WSM approach, had some limitations. The first limitation was the pairwise comparison of criteria against the tools. This process would have been efficient if a smaller number of tools were used for the study. Considering the 14 listed tools, however, it was a difficult process to manage the pairwise comparison. Again, it was observed that the use of AHP prevented the dynamic nature of weight assignment which happened as user's responded to the criteria-based questions.

For AHP, the weights were predetermined with the pairwise comparison as shown in *Table 27* and those weights are used in the mathematical model to produce the best decision out of the available alternatives. It was observed that, for the two tests (*Tables 28 and 29)* done using the AHP method, there was no change in the score since the weights of each criterion had already been determined in the pairwise comparison. Unlike the WSM model approach where the weight for the criteria is derived from the user's responses to the criteria-based questions, the AHP weights were already determined.

From the tests done, the weights and rankings of tools recommended with the AHP method remained the same after making some changes to the user's response. This is because the user's responses do not determine the criteria weights which is predefined by the pairwise comparison. From the results plotted in *Figure 30*, it is observed that the same set of responses for WSM gave different scores and rankings of the tools whereas the AHP method remained the same.

In comparison to the WSM approach, the need for tools recommendation based on user's feedback was necessary hence, AHP MCDM method as used by other researchers could not be recommended for this study. If AHP is to be used, then it must be used in conjunction with another MCDM method, like the TOPSIS as implemented by Abdulwareth et. al in their study [30], to allow for the dynamic assignment of weight to user preferences.

4.6.2 Manual Selection Process Comparison

The manual selection process is one main method used by many in the software testing industry. From the literature reviews done for this study, most of the tools evaluated were done manually in a table without any underlying way for choosing the right testing tools. Users often were presented with a comparison of tools from which they had to do the tedious work of manually evaluating which tool best fitted the type of test they wanted to perform.

Comparing this process with the method of using the WSM approach developed into a Python application. The selection process is done within minutes. Not only that, the tools are not manually evaluated by the user but the application that runs the WSM. This makes the selection process efficient compared to spending hours unending to check and evaluate the tool for your platform to test.

4.7 Limitations and Constraints

Despite the fact that the WSM approach had numerous advantages because of the aim of this study, some limitations and constraints were faced. This section outlines the various limitations and constraints.

The main limitation of the methodology utilised in this study was the use of a predefined set of testing tools in implementing the Python program. Fourteen (14) testing tools were used for the implementation which somewhat limited the scope of tools in the database to be used for the evaluation. This brought about the constraint of selecting the best automated tools from the available list of tools. This is the reason, the web version developed to allow for more available tools for selection.

Another limitation was on the AI-linked web-version. The testing tools could not be assigned pre-defined performance ratings. This is mainly because already existing AI models have the information on the myriad of test automation tools available on the market. Considering the variety of testing tools on the market, doing the comparison to get the performance ratings would have made the list finite just like it was for the Python code implementation.

One limitation of this research is the information gathering process done through the surveys. Due to the technical nature of this study, the survey participants had to be either software testers, quality assurance engineers or decision makers in the software testing industry. Getting these individuals to share their opinions and validate the tool developed was quite difficult hence limiting the sample space that partook in the survey.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Introduction

This research set out to develop a systematic and objective method for selecting the most appropriate automated software testing tool. This was based on the specific needs and requirements of the user or company. The findings of the research work are summarised in this chapter. Future work about software automation tool selection is also discussed along with some major conclusions and recommendations.

5.2 Major Findings of the Research

Through an extensive literature review and surveys, the key criteria that can be considered when evaluating and selecting automated testing tools were identified. Building on the widely used Weighted Sum Model (WSM) approach to multi-criteria decision making, this study proposed a structured framework that allows organisations to weight the relative importance of each evaluation criteria based on their unique priorities. By applying this WSM-based methodology, decision makers can methodically assess and rank available testing tools to determine the optimal solution.

To demonstrate the practical application of this framework, a Python application was developed with a case study evaluation of fourteen (14) leading automated testing tools including Selenium, Appium, Espresso, and TestComplete. The successful implementation of this process underscores the value and reliability of the WSM approach in supporting informed, justified decisions when selecting critical software testing solutions.

The key findings from this research include:

1. Identification of the critical evaluation criteria for automated software testing tools, thus: Ease of Use, Programming Skills, Scalability and Performance, Flexibility and Customization, Cost and Licensing, Vendor Stability and Product Support, and Reviews and Recommendations
2. The need for the separation of platform to test and type of testing required from the list of critical criteria required for automated testing tool selection. Previous work done usually added these two as criteria for evaluations. However, the filtering of tools into the two levels thus, platform to test and type of testing plays a critical role in the

evaluation of the software testing tools. This made the evaluation process for data driven and priority-based focussing only on the tools filtered based on the platform and type of test to be done.

3. Development of a structured WSM-based framework for systematically assessing and ranking automated testing tools based on the defined criteria and user's specific needs or teasing requirements. The use of the user's responses to assign weights to the different criteria identified made the decision-making process and selection of automated testing tools efficient.
4. Validation of the WSM-based approach as a reliable, data-driven method for making informed decisions on critical decisions like automated software testing tool selections. In this study, it was seen that the WSM method used as applied in this study made the multi-criteria decision analysis more appropriate for this study especially because of the dynamic nature of weight assignments based on the user's responses to the criteria-based questions.
5. The finding of this research is efficiency in the use of the WSM-based approach developed into an application for automated testing tool selection. Compared to other existing manual selection processes, this was very efficient recommending testing tools within a minute of answering the criteria-based questions.

While the case study involving fourteen (14) leading automated testing tools demonstrates the robustness and efficiency of the WSM-based framework, the generalizability of these findings may be constrained. The chosen tools such as Selenium, Appium, Espresso, and TestComplete, are representative of popular industry solutions but do not capture the full spectrum of automation tools, particularly emerging or domain-specific alternatives. As a result, the current validation provides a strong proof of concept but cannot claim universal applicability.

5.3 Conclusions

In conclusion, the software testing tools selection process does not have to be a daunting task for software testers, quality assurance engineers and decision makers in the software industry.

This research has successfully developed and validated a structured, multi-criteria decision-making framework for selecting the most appropriate automated software testing tool. By applying the proposed WSM-based methodology, stakeholders can systematically evaluate testing tool solutions against their unique requirements and priorities.

The Python code implementation with the 14 selected tools, alongside the AI-linked web version with an expanded toolset, demonstrates the practical application and benefits of this approach. The structured evaluation process, coupled with the ability to weight criteria based on user-specific needs, enables more informed and justified selections of automated testing tools. This ensures that chosen solutions effectively support quality assurance requirements and contribute to the success of software development efforts.

Importantly, the findings of this study are consistent with prior research that identified ease of use, cost, performance, and vendor support as central decision factors in tool selection. However, this research extends the literature by demonstrating that platform and type of testing should serve as filtering mechanisms rather than evaluation criteria, a distinction not made in several earlier studies.

Furthermore, while previous work has typically focused on manual evaluation processes, this study confirms and advances existing approaches by offering an automated, WSM-based framework capable of delivering recommendations within minutes. In this way, the results both validate established findings and contribute novel insights that enhance the efficiency and adaptability of automated testing tool selection.

5.4 Thesis Contribution

The following are the contributions of this study:

1. Multi-Level Filtering: Introduction of a novel two-stage filtering mechanism (platform to test and type of testing) prior to applying the Weighted Sum Model (WSM). Unlike previous studies that treated these factors as evaluation criteria, this methodological shift enhances precision by narrowing the evaluation space before applying the seven identified criteria. This represents a significant refinement of traditional MCDM approaches in the context of automated testing tool selection.
2. Preference-Driven Weighting: Incorporation of decision-maker preferences as dynamic weights in the WSM, ensuring that recommendations are directly aligned with specific organisational requirements and priorities.
3. Structured WSM-Based Framework: Development of a systematic, replicable methodology for evaluating and ranking automated testing tools, which addresses a long-standing gap in the literature on tool selection.

4. Practical Application & Validation: Demonstration of the framework's practical utility through both a Python-based implementation and an AI-linked web application, which can be readily adopted by organisations.
5. Empirical Validation of WSM Reliability: Confirmation of the WSM method's practicality and robustness for this domain, based on findings from surveys, implementation, and case study evaluation of 14 leading tools.

Collectively, these contributions extend existing research by providing not only a structured decision-making framework but also a methodological innovation that refines the traditional use of WSM in software engineering contexts.

5.5 Observations and Limitations

While the proposed WSM-based framework has been shown to be an effective tool for automated testing tool selection, there are a few observations and limitations to note:

1. The framework relies on the availability of accurate, comprehensive data on the capabilities of automated testing tools used for the evaluation. For the applications developed, the tools database depends on the accuracy of the information on each of the tools used for the evaluation process.
2. The weighting of evaluation criteria is subjective, as per the aim of this study, to have the user responses used as the weights for the criteria. Due to this, the results of tools recommended may vary across different organisations based on their unique priorities and constraints.
3. The Python code implementation focused on a specific set of fourteen testing tools; the framework can be validated against a broader range of solutions apart from the link to the AI model to give a countless number of testing tools for the evaluation process.
4. This research did not explore the potential integration of the WSM framework with other decision support tools or techniques. This could be considered in future studies.

5.6 Recommendations

Based on the findings and observations from this research, the following recommendations were provided:

1. Adoption of the Framework: Software testing professionals and decision-makers are encouraged to adopt the proposed WSM-based framework, or similar MCDM

approaches, as a structured, transparent method for selecting automated software testing tools. This ensures efficiency, cost-effectiveness, and alignment with organisational needs.

2. Technical Expansion: While this study employed Python for implementation, future adaptations should consider expanding the framework into widely used stacks such as Java or React, to increase accessibility and integration with existing software development environments.
3. Validation Across Contexts: To strengthen generalizability, the framework should be tested with a larger set of automated testing tools and across diverse groups of software testing professionals, including those from different industries and organisation sizes.
4. Refinement of Criteria: Combined criteria such as "flexibility and customization" should be decomposed into their constituent aspects, allowing decision-makers to prioritise more precisely according to their testing needs.

The following are some areas that can be considered for future work.

1. Machine Learning-Enhanced Selection: Explore the integration of learning algorithms to dynamically adjust tool recommendations based on accumulated user feedback and historical selections.
2. API and Platform Integration: Develop APIs to embed the tool selection framework into popular DevOps and CI/CD platforms, ensuring seamless adoption in real-world workflows.
3. Collaborative Decision Support: Extend the framework to support group decision-making, enabling multiple stakeholders to contribute weights, preferences, and feedback for more balanced selections.

REFERENCES

[1] B. Boehm, 'A view of 20th and 21st century software engineering', in *Proceedings of the 28th international conference on Software engineering*, Shanghai China: ACM, May 2006, pp. 12–29. doi: 10.1145/1134285.1134288.

[2] A. Bertolino, 'Software Testing Research: Achievements, Challenges, Dreams', in *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA: IEEE, May 2007, pp. 85–103. doi: 10.1109/FOSE.2007.25.

[3] V. Garousi, M. Felderer, M. Kuhrmann, K. Herkiloğlu, and S. Eldh, 'Exploring the industry's challenges in software testing: An empirical study', *J Software Evolu Process*, vol. 32, no. 8, p. e2251, Aug. 2020, doi: 10.1002/smr.2251.

[4] 'Software testing and analysis: process, principles, and techniques', *Choice Reviews Online*, vol. 46, no. 02, pp. 46-0935-46–0935, Oct. 2008, doi: 10.5860/CHOICE.46-0935.

[5] R. Charette, 'IEEE Spectrum: Why Software Fails', Aug. 2006, Accessed: Sep. 28, 2023. [Online]. Available: http://www.spectrum.ieee.org/print/1685

[6] V. Garousi and M. V. Mäntylä, 'When and what to automate in software testing? A multi-vocal literature review', *Information and Software Technology*, vol. 76, pp. 92–117, Aug. 2016, doi: 10.1016/j.infsof.2016.04.015.

[7] G. J. Myers, T. Badgett, and C. Sandler, 'The Art of Software Testing', *John Wiley & Sons, Inc., Hoboken, New Jersey*, 2004.

[8] M. Cinelli, M. Kadziński, M. Gonzalez, and R. Słowiński, 'How to support the application of multiple criteria decision analysis? Let us start with a comprehensive taxonomy', *Omega*, vol. 96, p. 102261, Oct. 2020, doi: 10.1016/j.omega.2020.102261.

[9] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, K. Petersen, and M. V. Mantyla, 'Benefits and limitations of automated software testing: Systematic literature review and practitioner survey', in *2012 7th International Workshop on Automation of Software Test (AST)*, Zurich, Switzerland: IEEE, Jun. 2012, pp. 36–42. doi: 10.1109/IWAST.2012.6228988.

[10] T. Varma, 'Automated software testing: introduction, management and performance', *SIGSOFT Softw. Eng. Notes*, vol. 25, no. 3, pp. 65–65, May 2000, doi: 10.1145/505863.505890.

[11] H. Bajaj, 'Viewpoint Choosing the Right Automation Tool And Framework Is Critical To Project Success', 2018.

[12] E. Borjesson and R. Feldt, 'Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry', in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, Montreal, QC, Canada: IEEE, Apr. 2012, pp. 350–359. doi: 10.1109/ICST.2012.115.

[13] M. E. Salari, E. Paul Enoiu, W. Afzal, and C. Seceleanu, 'Choosing a Test Automation Framework for Programmable Logic Controllers in CODESYS Development Environment', in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Valencia, Spain: IEEE, Apr. 2022, pp. 277–284. doi: 10.1109/ICSTW55395.2022.00055.

[14] A. Bhanushali, 'Ensuring Software Quality Through Effective Quality Assurance Testing: Best Practices and Case Studies', *Vol. No.*, no. 26, 2023.

[15] M. Tuteja and G. Dubey, 'A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models', vol. 2, no. 3, 2012.

[16] N. Martens, 'The impact of non-functional requirements on project success', Apr. 2011.

[17] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat, 'Classification of Software Testing Tools Based on the Software Testing Methods', in *2009 Second International Conference on Computer and Electrical Engineering*, Dubai, UAE: IEEE, 2009, pp. 229–233. doi: 10.1109/ICCEE.2009.9.

[18] C. J. Shingadiya and H. H. Patel, 'Modified Test Case Generation Using Testing Techniques', 2016.

[19] H. Kaur and D. G. Gupta, 'Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete', vol. 3, no. 5, 2013.

[20] K. Shaukat, U. Shaukat, F. Feroz, S. Kayani, and A. Akbar, 'Taxonomy of Automated Software Testing Tools', *International Journal of Computer Science and Innovation*, vol. Vol. 2015, no. 1, pp. 7–18, 2015.

[21] R. Torkar, *Towards automated software testing: techniques, classifications and frameworks*. Karlskrona: Blekinge Institute of Technology, 2006.

[22] M. Albarka Umar and C. Zhanfang, 'A Study of Automated Software Testing: Automation Tools and Frameworks', vol. Vol. 8, Nov. 2019.

[23] G. K. Saha, 'Understanding software testing concepts', *Ubiquity*, vol. 2008, no. February, pp. 1–1, Feb. 2008, doi: 10.1145/1361367.1348484.

[24] D. L. Asfaw, 'Benefits of Automated Testing Over Manual Testing', *International Journal of Innovative Resear ch in Infor mation Security*, vol. 2, no. 1, 2015.

[25] R. Mischke, K. Schaffert, D. Schneider, and A. Weinert, 'Automated and manual testing as part of the research software development process of RCE', 2022, *arXiv*. doi: 10.48550/ARXIV.2204.05600.

[26] D. Guan, 'Manual to Automated Testing', *Victoria University of Wellington*, Oct. 2014.

[27] D. Ateşoğulları and A. Mishra, 'Automation Testing Tools: A Comparative View', vol. 12, 2020.

[28] K. Naik and P. Tripathy, *Software testing and quality assurance: theory and practice*. Hoboken, N.J: John Wiley & Sons, 2008.

[29] N. Kryvinska and M. Greguš, Eds., *Data-Centric Business and Applications: Evolvements in Business Information Processing and Management—Volume 1*, vol. 20. in Lecture Notes on Data Engineering and Communications Technologies, vol. 20. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-319-94117-2.

[30] A. J. Abdulwareth and A. A. Al-Shargabi, 'Toward a Multi-Criteria Framework for Selecting Software Testing Tools', *IEEE Access*, vol. 9, pp. 158872–158891, 2021, doi: 10.1109/ACCESS.2021.3128071.

[31] N. Gogna, 'Study of Browser Based Automated Test Tools WATIR and Selenium', *IJIET*, vol. 4, no. 4, pp. 336–339, 2014, doi: 10.7763/IJIET. 2014.V4.425.

[32] R. Linner, 'The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed our Culture [Book Review]', *IEEE Technol. Soc. Mag.*, vol. 27, no. 2, pp. 61–64, 2008, doi: 10.1109/MTAS.2008.4538977.

[33] M. Olan and R. S. College, 'Unit Testing: Test Early, Test Often', 2003.

[34] K. Beck, *Test-driven development: by example*, 20. printing. in The Addison-Wesley signature series. Boston: Addison-Wesley, 2015.

[35] A. Holmes and M. Kellogg, 'Automating Functional Tests Using Selenium', in *AGILE 2006 (AGILE'06)*, Minneapolis, MN, USA: IEEE, 2006, pp. 270–275. doi: 10.1109/AGILE.2006.19.

[36] E. N. Budacu, 'Automating User Experience Testing in Mobile Development Teams', presented at the 18th International Conference on INFORMATICS in ECONOMY. Education, Research and Business Technologies, May 2019, pp. 333–338. doi: 10.12948/ie2019.05.05.

[37] W. M. Gamage, 'Test Automation Practices and Tools For Sustainable Software Development', *Lappeenranta–Lahti University of Technology LUT*, 2023.

[38] Rohit Khankhoje, 'An In-Depth Review of Test Automation Frameworks: Types and Trade-offs', *IJARSCT*, pp. 55–64, Oct. 2023, doi: 10.48175/IJARSCT-13108.

[39] Ms. Monika Gupta and Dr. R. K. Bathla, 'Comparative Study of Software Testing Technique using Manually and Automated Way', *IJSRST*, pp. 384–394, Dec. 2022, doi: 10.32628/IJSRST229657.

[40] E. Engström and P. Runeson, 'Software product line testing – A systematic mapping study', *Information and Software Technology*, vol. 53, no. 1, pp. 2–13, Jan. 2011, doi: 10.1016/j.infsof.2010.05.011.

[41] S. Berner, R. Weber, and R. K. Keller, 'Observations and Lessons Learned from Automated Testing', *ACM 1-58113-963-2/05/0005*, May 2005.

[42] S. Bucur, V. Ureche, C. Zamfir, and G. Candea, 'Parallel symbolic execution for automated real-world software testing', in *Proceedings of the sixth conference on Computer systems*, Salzburg Austria: ACM, Apr. 2011, pp. 183–198. doi: 10.1145/1966445.1966463.

[43] J. Kasurinen, O. Taipale, and K. Smolander, 'Software Test Automation in Practice: Empirical Observations', *Advances in Software Engineering*, vol. 2010, pp. 1–18, Feb. 2010, doi: 10.1155/2010/620836.

[44] R. Ramler, W. Putschögl, and D. Winkler, 'Automated testing of industrial automation software: practical receipts and lessons learned', in *Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, Hyderabad India: ACM, May 2014, pp. 7–16. doi: 10.1145/2593783.2593788.

[45] M. Böhme and S. Paul, 'On the efficiency of automated testing', in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong China: ACM, Nov. 2014, pp. 632–642. doi: 10.1145/2635868.2635923.

[46] R. Haas, R. Nömmer, E. Juergens, and S. Apel, 'Optimization of Automated and Manual Software Tests in Industrial Practice: A Survey and Historical Analysis', *IIEEE Trans. Software Eng.*, vol. 50, no. 8, pp. 2005–2020, Aug. 2024, doi: 10.1109/TSE.2024.3418191.

[47] P. Raulamo-Jurvanen, M. Mäntylä, and V. Garousi, 'Choosing the right test automation tool: A grey literature review of practitioner sources', in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jun. 2017, pp. 21–30. doi: 10.1145/3084226.3084252.

[48] Oscar Danilo Gavilánez Alvarez *et al.*, 'Comparative Analysis of Cross-Platform Frameworks', *JNS*, vol. 33, May 2023, doi: 10.59670/jns. v33i.874.

[49] S. Hotomski, E. Ben Charrada, and M. Glinz, 'Keeping Evolving Requirements and Acceptance Tests Aligned with Automatically Generated Guidance', in *Requirements Engineering: Foundation for Software Quality*, vol. 10753, E. Kamsties, J. Horkoff, and F. Dalpiaz, Eds., in Lecture Notes in Computer Science, vol. 10753., Cham: Springer International Publishing, 2018, pp. 247–264. doi: 10.1007/978-3-319-77243-1_15.

[50] N. Perera, D. I. D. Silva, C. Serasinghe, M. P. Gunathilake, S. Perera, and D. Samarasinghe, 'Examining the Role of Software Maintenance in Ensuring Software Quality', May 01, 2023. doi: 10.22541/au.168296460.09065818/v1.

[51] S. S. Murugiah, 'Adult Learning Theories and their Application in Selecting the Functionality of Synchronous Learning Tools', *Columbia University, USA*, 2005.

[52] E. Triantaphyllou, *Multi-Criteria Decision-Making Methods: A Comparative Study*, vol. 44. in Applied Optimization, vol. 44. Boston, MA: Springer US, 2000. doi: 10.1007/978-1-4757-3157-6.

[53] Biju Patnaik University of Technology (BPUT), Rourkela, Odisha, India, S. K. Sahoo, S. S. Goswami, and Biju Patnaik University of Technology (BPUT), Rourkela, Odisha, India, 'A Comprehensive Review of Multiple Criteria Decision-Making (MCDM) Methods: Advancements, Applications, and Future Directions', *Decis. Mak. Adv.*, vol. 1, no. 1, pp. 25–48, Dec. 2023, doi: 10.31181/dma1120237.

[54] H. Taherdoost and M. Madanchian, 'Multi-Criteria Decision Making (MCDM) Methods and Concepts', *Encyclopedia*, vol. 3, no. 1, pp. 77–87, Jan. 2023, doi: 10.3390/encyclopedia3010006.

[55] Dikky Suryadi, Warkianto Widjaja, Muchamad Sobri Sungkar, K. Kraugusteeliana, Iwan Adhicandra, and S. Sujito, 'Evaluating Location Alternatives for a New Manufacturing Plant using Weighted Sum Model Method', *J. Appl. Sci. Eng. Technol. Educ.*, vol. 5, no. 1, pp. 46–51, Apr. 2023, doi: 10.35877/454RI.asci1661.

[56] S. Rehman and S. A., 'Multi-Criteria Wind Turbine Selection using Weighted Sum Approach', *ijacsa*, vol. 8, no. 6, 2017, doi: 10.14569/IJACSA.2017.080616.

[57] M. R. Radhika and D. P. Rengarajan, 'Farmers' Views on Crop Insurance And Its Benefits Using Weighted Sum Model (Wsm)', 2024, doi: 10.53555/kuey. v30i5.4845.

[58] 'Future Research Opportunities Agricultural Sector Using Weighted sum method (WSM)', *cset*, vol. 1, no. 3, pp. 30–38, Sep. 2023, doi: 10.46632/cset/1/3/5.

[59] A. Paul, N. Shukla, S. K. Paul, and A. Trianni, 'Sustainable Supply Chain Management and Multi-Criteria Decision-Making Methods: A Systematic Review', *Sustainability*, vol. 13, no. 13, p. 7104, Jun. 2021, doi: 10.3390/su13137104.

[60] G. Yannis, A. Kopsacheili, A. Dragomanovits, and V. Petraki, 'State-of-the-art review on multi-criteria decision-making in the transport sector', *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 7, no. 4, pp. 413–431, Aug. 2020, doi: 10.1016/j.jtte.2020.05.005.

[61] A. Besimi, B. Cico, E. Pelivani, N. Macedonia, and C. Author, 'An Empirical Study Of User Interface Testing Tools', 2022. [Online]. Available: https://www.researchgate.net/publication/358913408

[62] M. A. Umar, 'A Study of Software Testing: Categories, Levels, Techniques, and Types', Oct. 30, 2023. doi: 10.36227/techrxiv. 12578714.v1.

[63] S. Izzat and N. N. Saleem, 'Software Testing Techniques and Tools: A Review', *JES*, vol. 32, no. 2, pp. 31–40, Jun. 2023, doi: 10.33899/edusj.2023.137480.1305.

[64] H. Prabhashi, M. Shafana, and M. Ahamed Sabani, *Comparative Analysis of Functional Test Automation Tools*. 2022.

[65] R. Ramli, A. R. Ahmad, and R. Ismail, 'Evaluating and Selecting Software Testing Tools: A Case Study', 2014, doi: 10.13140/2.1.4433.0247.

[66] V. Garousi and M. V. Mäntylä, 'A systematic literature review of literature reviews in software testing', *Information and Software Technology*, vol. 80, pp. 195–216, Dec. 2016, doi: 10.1016/j.infsof.2016.09.002.

[67] B. Potter and G. McGraw, 'Software security testing', *IEEE Secur. Privacy Mag.*, vol. 2, no. 5, pp. 81–85, Sep. 2004, doi: 10.1109/MSP.2004.84.

[68] C.-L. Hwang and K. Yoon, 'Methods for Multiple Attribute Decision Making', in *Multiple Attribute Decision Making*, vol. 186, in Lecture Notes in Economics and Mathematical Systems, vol. 186. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 58–191. doi: 10.1007/978-3-642-48318-9_3.

[69] A. Jahan, K. L. Edwards, and M. Bahraminasab, 'Multi-criteria decision-making for materials selection', in *Multi-criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design*, Elsevier, 2016, pp. 63–80. doi: 10.1016/b978-0-08-100536-1.00004-7.

[70] E. Triantaphyllou, *Applied Optimization - Multi-Criteria Decision-Making Methods_ A Comparative Study-Springer US (2000).pdf*, vol. 44. © Springer Science+Business Media Dordrecht, 2000.

APPENDICES

APPENDIX A

Validation Test Feedback Questions

1. What industry do you work in? Banking, Technology, Fintech, Other
2. Is this model and application useful for the software industry? Yes or No
3. Were the criteria considered relevant? Yes or No
4. Is using this model efficient for tool selection process? Yes or No
5. Is this model scalable? Yes or No
6. Is the model accurate? Yes or No
7. Is the application easy to use? Yes or No

APPENDIX B

I. Tools filtering and Criteria Based Questions for User Interface

Platform to test
What is the platform of the application under test? (e.g., web, mobile, API)?
Mobile Application
Web Application
API Services
Testing Requirements
What type of testing are you primarily performing? (e.g., functional, security, unit testing)?
Functional
Security
Performance
Unit
Integration
Ease of Use
How important is user-friendliness, GUI and ease of use for your team?
Very Important

Important
Neutral
Not very important
Not Important
Programming Skills
What level of programming skills does your team possess?
Beginner
Intermediate
Advanced
Scalability & Performance
Do you anticipate the need to scale your testing efforts in the future?
Yes
Maybe
No
Flexibility & Customization
Do you require the ability to customize and extend the tool's functionality?
Yes

Maybe
No
Licensing and Cost
Which of these licensing options will you prefer?
Open source (free)
Both
Commercial
Vendor Stability and Product Support
How important is the long-term viability and continued development of the tool?
Very Important
Important
Neutral
Not very important
Not Important
Reviews & Recommendations
How important is community support and user forums for troubleshooting?
Very Important

Important
Neutral
Not very important
Not Important

APPENDIX C

Python Code Implementation

```
# Defining the various inputs
def get_valid_input(prompt, valid_options):
  """User is prompted until a valid input is provided."""
  while True:
    response = input(prompt)
    if response in valid_options:
      return response
    else:
      print(f"Invalid input! Please choose one of the following: {', '.join(valid_options)}")
def get_user_input():
  print("This is an application that recommends testing tools based on your responses. Respond to the questions below to see the recommended tools for your automated testing.\n")
  platform_to_test = get_valid_input(
    "What is the platform of the application under test? (Enter one of the following responses: Web Application, Mobile Application, API Services): ",
    ["Web Application", "Mobile Application", "API Services"])
  testing_requirements = get_valid_input(
    "What type of testing are you primarily performing? (Enter one of the following responses: Functional, Security, Performance, Integration, Unit): ",
    ["Functional", "Security", "Performance", "Integration", "Unit"])
  ease_of_use = get_valid_input(
    "How important is user-friendliness, GUI and ease of use for your team? (Enter one of the following responses: Very Important, Important, Neutral, Not very important, Not Important): ",
```

["Very Important", "Important", "Neutral", "Not very important", "Not Important"])

programming_skills = get_valid_input(

"What level of programming skills does your team possess? (Enter one of the following responses: Beginner, Intermediate, Advanced): ",

["Beginner", "Intermediate", "Advanced"])

scalability_performance = get_valid_input(

"Do you anticipate the need to scale your testing efforts in the future? (Enter one of the following responses: Yes, Maybe, No): ",

["Yes", "Maybe", "No"])

flexibility_customization = get_valid_input(

"Do you require the ability to customise and extend the tool's functionality? (Enter one of the following responses: Yes, Maybe, No): ",

["Yes", "Maybe", "No"])

licensing_cost = get_valid_input(

"Which of these licensing options will you prefer? (Enter one of the following responses: Open source (free), Both, Commercial): ",

["Open source (free)", "Both", "Commercial"])

vendor_stability_roadmap = get_valid_input(

"How important is the long-term viability and continued development of the tool? (Enter one of the following responses: Very Important, Neutral, Not Important): ",

["Very Important", "Neutral", "Not Important"])

reviews_recommendations = get_valid_input(

"How important is community support and user forums for troubleshooting? (Enter one of the following responses: Very Important, Important, Neutral, Not very important): ",

["Very Important", "Important", "Neutral", "Not very important"])

return {

"Ease of Use": ease_of_use,

```
        "Programming Skills": programming_skills,
        "Scalability & Performance": scalability_performance,
        "Flexibility & Customization": flexibility_customization,
        "Licensing and Cost": licensing_cost,
        "Vendor Stability and Product Support": vendor_stability_roadmap,
        "Reviews & Recommendations": reviews_recommendations,
        "Platform to Test": platform_to_test,
        "Testing Requirements": testing_requirements
    }
def get_user_weights(user_responses, criteria_weights):
    user_weights = {}
    for criterion, response in user_responses.items():
        if criterion in criteria_weights:
            try:
                user_weights[criterion] = criteria_weights[criterion][response]
            except KeyError:
                print(f"Warning: The response '{response}' is not valid for '{criterion}'. It will be
skipped.")
    return user_weights
def filter_tools(tools, platform_to_test, testing_requirements):
    if isinstance(platform_to_test, str):
        platform_to_test = [platform_to_test]
    if isinstance(testing_requirements, str):
        testing_requirements = [testing_requirements]
    filtered_tools = []
    for tool in tools:
```

```
        if any(platform in tool["Platform"] for platform in platform_to_test) and \
                any(requirement in tool["Testing Requirement"] for requirement in testing_requirements):
            filtered_tools.append(tool)
    return filtered_tools
def calculate_tool_scores(filtered_tool_names, user_weights, ratings):
    tool_scores = {}
    for tool in filtered_tool_names:
        tool_scores[tool] = sum([
            user_weights.get("Ease of Use", 0) * ratings[tool][0],
            user_weights.get("Programming Skills", 0) * ratings[tool][1],
            user_weights.get("Scalability & Performance", 0) * ratings[tool][2],
            user_weights.get("Flexibility & Customization", 0) * ratings[tool][3],
            user_weights.get("Licensing and Cost", 0) * ratings[tool][4],
            user_weights.get("Vendor Stability and Product Support", 0) * ratings[tool][5],
            user_weights.get("Reviews & Recommendations", 0) * ratings[tool][6],
        ])
    return tool_scores
def get_top_tools(tool_scores, top_n=3):
    sorted_tools = sorted(tool_scores.items(), key=lambda x: x[1], reverse=True)
    return sorted_tools[:top_n]
def normalize_weights(weights):
    total = sum(weights.values())
    if total > 0:
        return {k: v / total for k, v in weights.items()}
    return weights
```

```
def weighted_sum_calculation(user_responses):
    # Criteria weights
    criteria_weights = {
        "Ease of Use": {
            "Very Important": 1.0,
            "Important": 0.8,
            "Neutral": 0.6,
            "Not very important": 0.4,
            "Not Important": 0.2,
        },
        "Programming Skills": {
            "Beginner": 1.0,
            "Intermediate": 0.8,
            "Advanced": 0.6,
        },
        "Scalability & Performance": {
            "Yes": 1.0,
            "Maybe": 0.8,
            "No": 0.6,
        },
        "Flexibility & Customization": {
            "Yes": 1.0,
            "Maybe": 0.8,
            "No": 0.6,
        },
        "Licensing and Cost": {
```

```
        "Open source (free)": 1.0,
        "Both": 0.8,
        "Commercial": 0.6,
    },
    "Vendor Stability and Product Support": {
        "Very Important": 1.0,
        "Important": 0.8,
        "Neutral": 0.6,
        "Not very important": 0.4,
        "Not Important": 0.2,
    },
    "Reviews & Recommendations": {
        "Very Important": 1.0,
        "Important": 0.8,
        "Neutral": 0.6,
        "Not very important": 0.4,
        "Not Important": 0.2,
    },
}
# Apply user responses to criteria weights
user_weights = get_user_weights(user_responses, criteria_weights)
# Normalising the user weights
normalized_weights = normalize_weights(user_weights)


# Extract platforms and testing requirements from user input
platform_to_test = user_responses["Platform to Test"]
```

```
testing_requirements = user_responses["Testing Requirements"]

# Tools database with their respective types of tests

tools = [

    {"Tool": "Appium", "Platform": ["Web Application", "Mobile Application"],

     "Testing Requirement": ["Functional", "Performance", "Integration"]},

    {"Tool": "Espresso", "Platform": ["Mobile Application"],

     "Testing Requirement": ["Functional", "Performance", "Unit"]},

    {"Tool": "XCUITest", "Platform": ["Mobile Application"],

     "Testing Requirement": ["Functional", "Performance", "Unit"]},

    {"Tool": "TestComplete", "Platform": ["Web Application", "Mobile Application", "API
Services"],

     "Testing Requirement": ["Functional", "Security", "Performance", "Unit",
"Integration"]},

    {"Tool": "Postman", "Platform": ["API Services"],

     "Testing Requirement": ["Functional", "Performance", "Integration"]},

    {"Tool": "SoapUI", "Platform": ["API Services"],

     "Testing Requirement": ["Functional", "Security", "Performance"]},

    {"Tool": "REST Assured", "Platform": ["API Services"], "Testing Requirement":
["Functional", "Performance"]},

    {"Tool": "JMeter", "Platform": ["Web Application", "API Services"],

     "Testing Requirement": ["Functional", "Performance", "Integration"]},

    {"Tool": "Selenium", "Platform": ["Web Application"],

     "Testing Requirement": ["Functional", "Integration", "Unit"]},

    {"Tool": "Cypress", "Platform": ["Web Application"], "Testing Requirement":
["Functional", "Integration"]},

    {"Tool": "Puppeteer", "Platform": ["Web Application"], "Testing Requirement":
["Functional", "Performance"]},
```

{"Tool": "TestCafe", "Platform": ["Web Application"], "Testing Requirement": ["Functional"]},

{"Tool": "Burp Suite", "Platform": ["Web Application"], "Testing Requirement": ["Security"]},

]

Filter tools based on user input

filtered_tools = filter_tools(tools, platform_to_test, testing_requirements)

Get the names of the filtered tools

filtered_tool_names = [tool["Tool"] for tool in filtered_tools]

Ratings (this should ideally be gathered from a reliable source or calculated based on criteria)

ratings = {

"Appium": [0.9, 0.8, 0.8, 0.7, 0.6, 0.9, 0.8],

"Espresso": [0.9, 0.9, 0.7, 0.6, 0.5, 0.8, 0.7],

"XCUITest": [0.8, 0.8, 0.6, 0.5, 0.6, 0.7, 0.7],

"TestComplete": [0.9, 0.8, 0.9, 0.8, 0.6, 0.9, 0.8],

"Postman": [0.9, 0.7, 0.8, 0.7, 0.6, 0.7, 0.8],

"SoapUI": [0.8, 0.8, 0.9, 0.9, 0.7, 0.8, 0.9],

"REST Assured": [0.8, 0.8, 0.8, 0.7, 0.6, 0.7, 0.8],

"JMeter": [0.8, 0.8, 0.8, 0.9, 0.6, 0.8, 0.9],

"Selenium": [0.9, 0.9, 0.9, 0.8, 0.6, 0.8, 0.8],

"Cypress": [0.9, 0.7, 0.6, 0.7, 0.6, 0.7, 0.8],

"Puppeteer": [0.8, 0.6, 0.5, 0.6, 0.5, 0.7, 0.7],

"TestCafe": [0.7, 0.6, 0.6, 0.5, 0.5, 0.7, 0.6],

"Burp Suite": [0.9, 0.8, 0.9, 0.9, 0.7, 0.8, 0.9],

}

Calculate tool scores

```
    tool_scores = calculate_tool_scores(filtered_tool_names, normalized_weights, ratings)
    # Get the top 3 tools based on scores
    top_tools = get_top_tools(tool_scores)
    return top_tools
if __name__ == "__main__":
    user_responses = get_user_input()
    top_tools = weighted_sum_calculation(user_responses)
    print("\nBased on your responses, the top recommended testing tools are:")
    for tool, score in top_tools:
        print(f"{tool}: Score = {score:.2f}")
```

APPENDIX D

I. Html Code For Web Application Version

```
<!DOCTYPE html>
<html>
<head>
  <title>Automated Testing Tool Selector</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-colour: #f5f5f5d5;
    }
    .container {
      max-width: 700px;
      margin: 0 auto;
      padding: 20px;
      background-colour: rgb(237, 216, 216);
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    #responseContainer {
      padding: 20px;
      border: 1px solid #ccc;
      border-radius: 8px;
      margin: 20px 0;
      background-colour: #f9f9f9;
      font-family: Arial, sans-serif;
```

```
}
.responseTitle {
    font-weight: bold;
    font-size: 24px;
    margin-bottom: 20px;
}
.responseContent {
    font-size: 18px;
    line-height: 1.5;
}
select {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
}
label {
    font-weight: bold;
}
h3 {
    font-style: italic;
    colour: #007BFF;
    /* Blue colour for section titles */
}
button {
    background-colour: #007BFF;
    /* Blue button background colour */
```

```
        colour: #fff;
        /* White button text colour */
        border: none;
        border-radius: 5px;
        padding: 10px 20px;
        font-size: 18px;
        cursor: pointer;
    }
    button:hover {
        background-colour: #0056b3;
        /* Darker blue on hover */
    }


    a {
        colour: #007BFF;
        /* Blue hyperlink text colour */
        text-decoration: none;
        margin-top: 10px;
        display: block;
    }
    a:hover {
        text-decoration: underline;
    }
  </style>
</head>
```

```
<body>
  <div class="container">
    <h1>Automated Testing Tool Selector</h1>
    <h5>This is an application that recommends testing tools based on your responses. Respond to the questions below to see the recommended tools for your
      automated testing.</h5>
    <form id="testingToolForm">

  <h3>Features and Functionality</h3>
      <label for="testPlatform">What platform are you testing? (e.g., web, mobile, API)</label>
      <select id="testPlatform">
        <option value="" disabled selected>Choose an option</option>
        <option value="Web Application">Web Application</option>
        <option value="Mobile Application">Mobile Application</option>
        <option value="API Services">API Services</option>
      </select>

  <h3>Testing Requirements</h3>
      <label for="typeOfTests">What type of testing are you primarily performing? (e.g., functional, security,
        unit testing)?</label>
      <select id="typeOfTests">
        <option value="" disabled selected>Choose an option</option>
        <option value="Functional Testing">Functional</option>
        <option value="Security Testing">Security</option>
        <option value="Performance Testing">Performance</option>
```

```
        <option value="Integration Testing">Integration</option>
        <option value="Unit Testing">Unit</option>
    </select>

    <h3>Ease of Use</h3>
    <label for="usability">How important is user-friendliness, GUI and ease of use for your team?</label>
    <select id="usability">
        <option value="" disabled selected>Choose an option</option>
        <option value="Very Important">Very Important</option>
        <option value="Important">Important</option>
        <option value="Neutral">Neutral</option>
        <option value="Not Very Important">Not Very Important</option>
        <option value="Not Important">Not Important</option>
    </select>

    <h3>Programming Skills</h3>
    <label for="proficiency">What level of programming skills does your team possess?</label>
    <select id="proficiency">
        <option value="" disabled selected>Choose an option</option>
        <option value="Yes">Beginner</option>
        <option value="No">Intermediate</option>
        <option value="No">Advanced</option>
    </select>
```

```
<h3>Scalability & Performance</h3>
<label for="scaling">Do you anticipate the need to scale your testing efforts in the future?</label>
<select id="scaling">
  <option value="" disabled selected>Choose an option</option>
  <option value="Yes">Yes</option>
  <option value="No">No</option>
</select>

<h3>Flexibility & Customization</h3>
<label for="customization">Do you require the ability to customise and extend the tool's functionality?</label>
<select id="customization">
  <option value="" disabled selected>Choose an option</option>
  <option value="Extensive Customization Needed">Extensive Customization Needed</option>
  <option value="Some Customization Needed">Some Customization Needed</option>
  <option value="Limited Customization Needed">Limited Customization Needed</option>
  <option value="No Customization Needed">No Customization Needed</option>
</select>

<h3>Licensing and Cost</h3>
<label for="openSourceCommercial">Which of these licensing options will you prefer?</label>
```

```
<select id="openSourceCommercial">
    <option value="" disabled selected>Choose an option</option>
    <option value="Open Source">Open Source</option>
    <option value="Commercial">Commercial</option>
    <option value="Both">Both</option>
</select>
<h3>Vendor Stability and Roadmap</h3>
<label for="longTermViability">How important is the long-term viability and continued development of the
    tool?</label>
<select id="longTermViability">
    <option value="" disabled selected>Choose an option</option>
    <option value="Very Important">Very Important</option>
    <option value="Important">Important</option>
    <option value="Neutral">Neutral</option>
    <option value="Not Very Important">Not Very Important</option>
    <option value="Concerned about Obsolescence">Concerned about Obsolescence</option>
</select>
<h3>Reviews & Recommendations</h3>
<label for="communitySupport">How important is community support and user forums for
    troubleshooting?</label>
<select id="communitySupport">
    <option value="" disabled selected>Choose an option</option>
    <option value="Very Important">Very Important</option>
    <option value="Important">Important</option>
```

```
            <option value="Neutral">Neutral</option>
            <option value="Not Very Important">Not Very Important</option>
            <option value="Not Important">Not Important</option>
        </select>
        <h4>Now that you are done entering your responses, click/tap the submit button below to see which tools will be recommended for you.</h4>
        <button type="submit">Submit</button>
        <div id="responseContainer"></div>
<script src="script.js"></script>
    </form>
  </div>
</body>
</html>
```

II. Javascript Code Implementation

```
document.getElementById('testingToolForm').addEventListener('submit', function (event) {
  event.preventDefault();  // This prevents the form from causing a page reload
    // Get all dropdown values
  const testPlatform = document.getElementById('testPlatform').value;
  const typeOfTests = document.getElementById('typeOfTests').value;
  const usability = document.getElementById('usability').value;
  const proficiency = document.getElementById('proficiency').value;
  const scaling = document.getElementById('scaling').value;
  const customization = document.getElementById('customization').value;
```

```
    const openSourceCommercial = document.getElementById('openSourceCommercial').value;

    const longTermViability = document.getElementById('longTermViability').value;

    const communitySupport = document.getElementById('communitySupport').value;


    // Check if any dropdown is not selected (value is an empty string "")

    if (!testPlatform || !typeOfTests || !usability || !proficiency || !scaling || !customization || !openSourceCommercial || !longTermViability || !communitySupport) {

        // Display feedback to the user

        document.getElementById('responseContainer').innerHTML = `

            <div class="responseTitle">Error:</div>

            <div class="responseContent">

                <p>Please make sure you select an option for all fields before submitting.</p>

            </div>

        `;

        return;  // Exit the function, preventing the form from being processed

    }


    // If all dropdowns are selected, show loading message

    const responseHTML = `

        <div class="responseTitle">Loading...</div>

    `;
```

```
document.getElementById('responseContainer').innerHTML = responseHTML;


// Construct the complete prompt
const completePrompt = `
Given the points below with their various answers, use the Weighted Sum Model multi-criteria method to recommend the best three testing tools to use.
Test Platform: ${testPlatform}
Type Of Tests: ${typeOfTests}
Usability: ${usability}
Proficiency: ${proficiency}
Scaling: ${scaling}
Customization: ${customization}
Open Source/Commercial: ${openSourceCommercial}
Long Term Viability: ${longTermViability}
Community Support: ${communitySupport}
`;
console.log(completePrompt);
// Request body
const requestBody = {
    max_tokens: 512,
    model: "mistralai/Mixtral-8x22B-Instruct-v0.1",
    prompt: completePrompt,
```

```
    prompt_format_string: "user{prompt}assistant",
    repetition_penalty: 1,
    repetitive_penalty: 1,
    request_type: "language-model-inference",
    temperature: 0.7,
    top_k: 50,
    top_p: 0.7,
    type: true
};
const ENDPOINT_URL = 'https://api.together.xyz/inference';
const API_KEY =
'7772f4eeaf9c5a6588071f168ec4c4fc54f8ec69ecdb8c73a831d90e3e0236f4';
// Send post request to API
fetch(ENDPOINT_URL, {
    method: 'POST',
    headers: {
        'Authorization': `Bearer ${API_KEY}`,
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(requestBody)
})
.then(response => response.json())
```

```
    .then(data => {
        // Handle the response data
        console.log(data);
        const responseText = data.output.choices[0].text;
        const paragraphs = responseText.split('\n\n');
        let responseHTML = '<div class="responseTitle">Response:</div><div class="responseContent">';
        paragraphs.forEach(paragraph => {
            responseHTML += `<p>${paragraph}</p>`;
        });
        responseHTML += '</div>';
        document.getElementById('responseContainer').innerHTML = responseHTML;
    })
    .catch(error => {
        console.error('Error:', error);
        document.getElementById('responseContainer').innerHTML = '<div class="responseTitle">Error</div><div class="responseContent"><p>There was an error processing your request. Please try again later.</p></div>';
    });
});
```