Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technology

Master's Thesis

A Comparative Study between Manual and Automation Methods & Tools for Software Testing

Vijayalakshmi Anandaiah

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Eng. Vijayalakshmi Anandaiah, B.E.

Global Information Security Management

Thesis title

A Comparative Study between Manual and Automation Methods & Tools for Software Testing

---

Objectives of thesis

The main objective of this thesis is to compare the effectiveness of manual testing and automation testing in terms of identifying defects and ensuring software quality.
The supportive goals are:
To analyze the benefits and drawbacks of each testing approach concerning resource utilization, time efficiency, and test coverage.
To explore and evaluate popular automation testing tools and frameworks to understand their suitability for different testing scenarios.
To identify the challenges and limitations associated with automation testing, including test case maintenance, test data management, and skill requirements for testers.
To provide real-world case studies illustrating the practical implementation of manual and automation testing in different software development environments.
To propose best practices for combining manual and automated testing to maximize testing efficiency and achieve optimal test coverage.
To highlight future trends in software testing, such as advancements in test automation, the integration of AI and ML, and the impact of Agile and DevOps on testing methodologies.

Methodology

The research design for this comparative study will be a mixed-method approach, incorporating both qualitative and quantitative methods. This will allow for a comprehensive analysis of the effectiveness and efficiency of manual and automated testing methods.

This thesis highlights the importance of automation and manual testing to cover software development's functionality, security, performance, and usability aspects. The project mainly compares manual and automated testing processes, methods, and techniques to produce defect-free products. It explains the Software Testing Life Cycle process, an integral part of the Software Development Life cycle. It describes tools and strategies implemented to overcome the drawbacks of manual and automation testing, respectively. It mainly explains how to write the test cases and execution of those test cases generally in both manual and automation testing. This will allow us to better understand testing concepts by exploring the innovation of tools and technologies incorporated in the IT industry.

This comprehensive comparative study helps us make informed decisions about when to use manual testing when to use automation, and how to optimize your testing efforts based on your specific project needs.

The proposed extent of the thesis

60 – 80 pages

Keywords

Functionality, Performance, GUI testing, Manual and Automation testing tools & techniques, SDLC,STLC, Selenium Webdriver, Test Cases, Quality Assurance

---

Recommended information sources

Chaudhary, Sarika. "Latest software testing tools and techniques: A review." International Journal 7.5 (2017).

Jan, S. Roohullah, et al. "An innovative approach to investigate various software testing techniques and strategies." International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Print ISSN 23951990 (2016).

Pelivani, Elis, and Betim Cico. "A comparative study of automation testing tools for web applications." 2021 10th Mediterranean Conference on Embedded Computing (MECO). IEEE, 2021.

Quadri, S. M. K., and Sheikh Umar Farooq. "Software testing–goals, principles, and limitations." International Journal of Computer Applications 6.9 (2010): 1.

Saha, Trina, and Rajesh Palit. "Practices of software testing techniques and tools in Bangladesh software industry." 2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE). IEEE, 2019.

Sawant, Khushboo, et al. "Implementation of selenium automation & report generation using selenium web driver & ATF." 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT). IEEE, 2021.

Sharma, Monika, and Rigzin Angmo. "Web based automation testing and tools." International Journal of Computer Science and Information Technologies 5.1 (2014): 908-912.

Sneha, Karuturi, and Gowda M. Malle. "Research on software testing techniques and software automation testing tools." 2017 international conference on energy, communication, data analytics and soft computing (ICECDS). IEEE, 2017.

Srivastava, Nishi, Ujjwal Kumar, and Pawan Singh. "Software and performance testing tools." Journal of Informatics Electrical and Electronics Engineering (JIEEE) 2.1 (2021): 1-12.

Umar, Mubarak Albarka, and Chen Zhanfang. "A comparative study of dynamic software testing techniques." International Journal of Advanced Networking and Applications 12.3 (2020): 4575-4584.

Expected date of thesis defence

2023/24 SS – PEF

The Diploma Thesis Supervisor

Ing. Václav Lohr, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 4. 9. 2023	Electronic approval: 3. 11. 2023
doc. Ing. Jiří Vaněk, Ph.D.	**doc. Ing. Tomáš Šubrt, Ph.D.**
Head of department	Dean

Prague on 01. 03. 2024

Declaration

I affirm that, I have independently completed my master’s thesis entitled "**A Comparative Study between Manual and Automation Methods & Tools for Software Testing**" and have solely utilized the references cited in the thesis. As the Author of the master’s thesis, I assert that it does not infringe upon any copyright laws.

In Prague on 27-03-2024

Vijaya

Vijayalakshmi Anandaiah

Acknowledgement

I would like to extend my sincere gratitude to Ing. Václav Lohr, Ph.D., my thesis advisor, for his invaluable guidance, unwavering assistance, and insightful feedback throughout this research project. As a result of the expertise and capabilities possessed by these individuals, the direction and quality of this thesis have been significantly enhanced.

It is my pleasure to extend my sincere thanks to my esteemed professors, dear classmates, and the dedicated academic staff of the Global Information Security Management (GISM) department at the Czech University of Life Sciences, Prague. During the past two years, their consistent support has played a significant role in shaping and enriching my academic journey. As a result of the encouragement, guidance, and shared knowledge, I have been able to deepen my understanding of the subject matter and have also created a collaborative and enriching learning environment. My sincere gratitude goes out to the entire academic community for their invaluable contributions to my educational journey, and I sincerely appreciate the collective efforts that have contributed to my personal growth and development.

Lastly, I wish to express my thanks to my family and friends for their understanding, perseverance, and support throughout my academic journey.

A Comparative Study between Manual and Automation Methods & Tools for Software Testing

Abstract

This thesis focuses on software testing methodologies, and manual versus automation testing is particularly compared. The main aim is to identify how each method detects defects and assures software quality. Using a research method, which combines qualitative and quantitative analyses, the aspects of resource utilization, time efficiency, and test coverage are examined to highlight the pros and cons of the manual and automated testing.

In addition, the performance of leading automation testing tools and frameworks is reviewed to establish their suitability in various testing contexts. The issues of automation testing, which include maintenance of test cases, data management, and necessary skill sets for testers, are discussed. The case studies from the real world provide knowledge on manual and automation testing implementation in different software development fields.

Taking from the results, suggestions are made for improving testing speed and obtaining a complete test coverage through the balanced use of both manual and automated testing approaches. Furthermore, future directions of the area of software testing, including the development of automated testing, the application of AI and ML, and the changing role of Agile and DevOps methodologies are considered.

In this respect, this thesis adds to the understanding of the software testing lifecycle within the wider context of software development by demanding respect for both manual and automation testing in guaranteeing software functionality, security, performance, and usability. It provides those in practice with practical ideas that let them choose correctly by requirements of their project.

Keywords: Functionality, Performance, GUI testing, Manual and Automation testing tools & techniques, SDLC, STLC, Selenium WebDriver, Test Cases, Quality Assurance.

Srovnávací studie mezi manuálními a automatizačními metodami a nástroji pro testování softwaru

Abstrakt

Tato práce se zaměřuje na metodiky testování softwaru a srovnává zejména manuální versus automatizační testování. Hlavním cílem je identifikovat, jak jednotlivé metody detekují defekty a zajišťují kvalitu softwaru. Pomocí výzkumné metody, která kombinuje kvalitativní a kvantitativní analýzy, jsou zkoumány aspekty využití zdrojů, časové efektivity a pokrytí testem, aby se zdůraznily výhody a nevýhody manuálního a automatizovaného testování.

Kromě toho je přezkoumán výkon předních nástrojů a rámců pro automatizaci testování, aby se zjistila jejich vhodnost v různých testovacích kontextech. Diskutovány jsou otázky automatizačního testování, které zahrnují údržbu testovacích případů, správu dat a nezbytné sady dovedností pro testery. Případové studie z reálného světa poskytují znalosti o implementaci manuálního a automatizačního testování v různých oblastech vývoje softwaru.

Na základě výsledků jsou předloženy návrhy na zlepšení rychlosti testování a získání úplného pokrytí testováním prostřednictvím vyváženého používání manuálních i automatických testovacích přístupů. Dále jsou zvažovány budoucí směry v oblasti testování softwaru, včetně vývoje automatizovaného testování, aplikace AI a ML a měnící se role metod Agile a DevOps.

V tomto ohledu tato práce přispívá k pochopení životního cyklu testování softwaru v širším kontextu vývoje softwaru tím, že vyžaduje respektování manuálního i automatizačního testování při zaručení funkčnosti, bezpečnosti, výkonu a použitelnosti softwaru. Praktikujícím poskytuje praktické nápady, které jim umožňují správně si vybrat podle požadavků jejich projektu.

Klíčová slova: Funkčnost, výkon, testování GUI, ruční a automatizační testovací nástroje a techniky, SDLC, STLC, Selenium WebDriver, testovací případy, zajištění kvality.

Table of Contents

1. Introduction

Software testing is a crucial step in the SDLC that guarantees the quality, dependability, and performance of the application. Before use, the application is carefully analyzed against specified requirements to find and eliminate any errors. In the past, human testers performed this process manually, carefully following test protocols and recording results. However, due to the complexity and proliferation of current software applications, manual testing has become progressively less sufficient. As a result, testing machines have become popular. This approach uses scripts and specialized tools to automatically execute the test case.

When it comes to quality assurance in the ever-changing world of software development, the choice of people and automated testing methods is important by comparing these two methods, this review aims to clarify their differences and similarities as well as their advantages and disadvantages. The traditional mainstay of quality assurance, manual testing, relies on human interaction to carefully navigate software systems, find bugs, and guarantee functionality because it is flexible, it can be consumed role for survey research, ad-hoc testing, and evaluation experience. This makes it extremely valuable in situations where human emotion and creativity are critical. However, scalability, reproducibility, and speed issues can arise with the manual approach, especially as software projects become larger and more sophisticated. (Singh 2023)

However, the advent of automated testing introduced a paradigm shift in the testing industry. Automated testing improves coverage, performance, and repeatability by using specialized tools and scripting to execute predefined test cases. Regression testing, performance testing, and repetitive tasks are some of the situations where this approach comes in handy. Teams can identify issues early in the development cycle and accelerate test execution. But because it can miss out on the richness of the user experience and requires a huge upfront investment to set up, automation isn’t a solution for everyone.

The human factor is one of the key elements in this comparative study. Manual testing to find minor faults and confirm the user experience relies on the sensory awareness, instinct, and domain knowledge of the tester. This approach works especially well for exploratory testing, as it allows human testers to simulate real-world scenarios, adapt to changing needs, and evaluate software from the users' perspective. It is critical to strike a balance between automatic and manual testing, since each approach has its own set of pros and cons. (Kumari, Chauhan, Vedpal 2018)

Scalability is important in the world of software testing, especially as projects become larger and more complex. Due to its resource consumption, manual testing can become cumbersome as the number of test cases and iterations increases. It can be difficult to manage and coordinate large groups of manual testers, which can lead to testing delays. Scalability issues are best addressed through automation because it can run multiple test cases in a short amount of time. Automated testing frameworks allow for parallel execution, helping teams manage large test suites faster and providing developers with timely feedback.

While automated testing increases performance, there are situations where manual testing—which involves human intervention—is necessary. Manual testing excels in functional testing, access testing, and situations that require subjective review. Testers can identify problems that automated scripts have missed, analyse the overall user experience, and even spot abnormalities visually. Furthermore, human intelligence and adaptability are essential for exploratory research, in which testers actively search the application for hidden bugs (Enoiu 2017)

The cost-effectiveness of the two test methods is an important aspect of the comparative analysis. Although labour-intensive, manual testing can be expensive for smaller companies with simple requirements. The time and effort required to develop scripts can stack up, as can the initial investment in automated testing tools and frameworks. However, automated testing is more economical in the long run when projects are large and test requirements are repetitive. Regression testing can be performed quickly and reliably, reducing overall testing effort and speeding delivery.

Effective software testing requires collaboration and communication between development teams. Collaboratively, where feedback is quick and relevant, manual testing encourages a direct relationship between developers and testers. Testers can communicate with developers in real time, generate comprehensive reports, and simulate on-site issues. While automated testing speeds up testing, it creates a barrier to separation between developers and testers. Balancing the rigor of automated testing with the detailed perspective that manual testing provides requires effective teamwork.

In summary, a comparison of software testing tools and methods manual and automated reveals a complex situation in which each method has its own advantages Unlike manual testing in conditions that require empirical research, diagnostic testing, and usability evaluation, especially in developing regression testing and performance testing. It is important to find an ideal mix of manual and automated testing, considering the size, complexity, and specialized testing requirements of the project. Knowing the pros and cons of manual and automated testing enables teams in an ever-evolving software development process to make informed decisions that ultimately result in better software products. (Daniel Sundmark 2019)

2. Objectives and Methodology

2.1 Main Objective

This thesis delves into the important issue of effective software testing. The main objective is to differentiate between two well-known testing methods, automation testing and manual testing. This study attempts to address the central issue: What is the best method for finding defects and guaranteeing high-quality software?

To achieve this primary objective, the thesis will explore several other objectives:

- To analyze the strengths and weaknesses of each testing approach in terms of resource utilization, time efficiency, and test coverage.
- To explore and evaluate popular automation testing tools and frameworks to understand their suitability for different testing scenarios.
- To identify the challenges and limitations associated with automation testing, including test case maintenance, test data management, and skill requirements for testers.
- To provide real-world case studies illustrating the practical implementation of manual and automation testing in different software development environments.
- To propose best practices for combining manual and automated testing to maximize testing efficiency and achieve optimal test coverage.
- To highlight future trends in software testing, such as advancements in test automation, the integration of AI and ML, and the impact of Agile and DevOps on testing methodologies.

2.2 Supportive Goals

To support this comparison of manual and automated testing techniques and technologies, the following supporting objectives are examined in further detail:

2.2.1 Resource Utilization, Time Efficiency, and Test Coverage

Evaluate the economic impact of humans and laboratories to quantify the impact on resources. Determine the number of human testers required for various testing scenarios of the resources required to maintain automation tools and scripts.

Time Efficiency Benchmarking:

Determine the time difference between typical test cases and execution using automation scripts. This will measure how much time automation saves and how it affects testing processes.

Test Coverage Analysis:

Examine how well automation and manual testing work together to achieve adequate test coverage. Examine how each method distinguishes between types of errors (e.g., functional vs. functional). Find out how the two methods can be combined to create a comprehensive test plan. (Hamilton 2024)

2.2.2 Evaluation of Automation Testing Tools and Frameworks

Functional Deep Dive:

Open major automation frameworks (like Cypress, Selenium, Katallon Studio) and tools (like Appium, Katallon Studio) and explore their capabilities in depth. Required features such as detectors, data-driven testing capabilities, reporting features and device integration are all part of this.

Comparative Analysis Matrix:

Create a comparative analysis matrix showing the advantages and disadvantages of different automation frameworks. Factors to consider include community support, scalability for large test suites, supported programming languages, ease of use, and learning curve.

Selection Criteria:

Provide software development teams with a decision-making process so they can choose the best automation solution for their unique testing needs. This design should take into account variables such as project complexity, funding availability, level of technical expertise, and type of application being tested (K. 2022)

2.2.3 Challenges and Limitations of Automation Testing

Script maintenance methods:

Discuss various methods used to handle the ongoing maintenance of automation scripts, such as automated regression testing tools, refactoring methods, and version control to assess script reliability, script clarity, or incompressibility never show -Look for best practices to reduce failure.

Non-experimental areas:

Look at tests that, such as exploratory testing, user interface (UI) testing, and testing that require humans to make judgments or measurements self-inadequate for automated testing. (Adil 2024)

2.2.4 Real-world Case Studies

Appropriate Case Analysis:

Identify real case studies where software development teams have successfully applied a balanced approach to both manual and automated testing. Prepare test cases, test methodologies and consumption tools role, and quantifiable results.

Measurable Outcomes:

Focus on the measurable results obtained in the case studies, such as increased test payment percentages, shorter test times, and error detection a it goes upwards.

Generalizable learning:

Draw conclusions from case studies that are relevant in experimental approaches. This may involve identifying the best ways to integrate humans and laboratories or adopting solutions to specific experimental problems. (Dilmegani 2024)

2.2.5 Best Practices for Manual and Automated Testing

Effective manual testing strategies:

Define best practices for manual testing. Attention should be paid to techniques such as exploratory testing, in which testers use their experience to build test cases on the fly, and functional testing, which focuses primarily on the user’s interaction with the application.

Creating automation scripts:

Provide documentation for reliable and maintainable creation of automation scripts. This includes best practices for writing reusable and modular code, how to use data-driven testing methods to improve scripts, and how to set up reporting and logging systems to analyse test results.

Collaborative Testing Approach:

Encourage collaboration in which automation and manual testing are viewed as complimentary capabilities of one common testing process rather than two separate entities. To ensure a seamless testing process, it includes developing seamless communication channels between manual and automated assessors. (Bot play automation 2022)

2.2.6 Future Trends in Software Testing

Software testing is a procedure that includes examining and confirming that each component of a software program is functioning in the acceptable manner. In order to do this, it may be necessary to assess the functionality of a software in addition to its accessibility, usability, and security characteristics. It can be necessary to test the program in several environments to guarantee it works with Windows, Linux, iOS, Android, and a plethora of other devices and operating systems. Smartphones, laptops, notepads, and desktop workstations are all examples of settings that fall under this category.

The process of testing software also involves deciding whether or not an application will be of use to a corporation in accomplishing its goals. Although UI is an abbreviation for "user interface," UX is an abbreviation for "user experience." For example, if a company that specializes in e-learning wants to launch a web-based application (one that is more user-friendly than the application they are now using), then the process of developing software will contain both of these concepts. Also known as KPIs, or Key Performance Indicators, for short. With the use of these key performance indicators (KPIs), the team will be able to evaluate the program's usability and accessibility. For example, they will be able to determine how easy it is to navigate the application and find the information or features that are required.(Rajesh Kumar Mishra 2017)

Functional testing, unit testing, performance testing, and regression testing are some of the several types of software testing that are available. There is a vast range of software testing, which encompasses a number of testing methods. The objective of every kind of examination is to evaluate a distinct group of elements and to provide a distinct collection of outcomes. As an example, unit testing is used to ensure that each and every component of a software program performs properly, while performance testing is used to examine how efficiently an application runs when it is exposed to various quantities of labor. Using the proper testing techniques in order to test the relevant criteria is very vital if you want to get the most out of the testing experience. This is because you want to get the most out of the testing experience.(Desikan 2006)

Integrating AI and Machine Learning:

There have been decades of progress in the subject of automation testing, which includes the construction of test scripts that can be repeated automatically. In recent years, this topic has received considerable attention. Nevertheless, despite the fact that this approach offers a number of benefits, such as the automation of testing scenarios that are repetitive, time-consuming, and prone to mistakes, a considerable amount of human work is still required from the users. In the first place, it is necessary for testers to manually update their test scripts whenever there is a modification made to an application software. This is due to the fact that the parameters, which are nothing more than a collection of actions performed by the user, could no longer correlate to the functioning of the software. This is an activity that takes a significant amount of time and is characterized by it being repetitive.

Because of the contributions made by artificial intelligence and machine learning, it is possible that the amount of manual labor that is associated with test automation technologies might be decreased. Through the provision of training data to a machine learning model, testers are able to automatically generate, audit, and carry out test cases. It is possible that logs, test cases, and documentation are included in this training data.(Drusinsky 2017)

The machine learning model will get cleverer as a result of frequent training. It will also become better able to grasp how the application program works and will be more able to determine what the user experience should be like. When this is finished, the machine learning model will be able to produce data that is more accurate and insightful, which developers may use to find solutions to issues and make adjustments. An additional benefit of machine learning models is that they may be trained to develop and run test cases that are representative of the most current version of an application software. This is a significant advantage. As a result, the problem of incompatibility between versions is resolved.

Impact of Test Design:

Discuss how these trends may affect the future of both human and automated testing. Consider how AI can automate many testing scenarios, thereby reducing the need for human testing of specific functionality. However, strategic decision-making and complex experimental situations may always require human skills.

Adapting to Change:

Explain how software teams can prepare for and adapt to these rapidly changing test industry trends.(Jackson-Barnes 2024)

Latest Trends in Software Testing:

Below is a complete study of the trends in software testing that are now occurring and those that are expected to occur in the near future. These patterns will have a big influence on the future. On the other hand, some of these tendencies are already being put into practice, while others are on the approach of becoming mainstream. It is useful to be aware of these patterns regardless of the conditions since it gives a method of knowing how different software development businesses test their applications, both in the present and in the future. This awareness may be valuable.

In light of this, if you are going to hire a software development team in the near future, then knowing this knowledge will be of great assistance to you in picking a team that puts a high focus on testing.

Robotic Process Automation (RPA):

RPA, which stands for robotic process automation, is a method that incorporates the use of software robots for the purpose of automating business procedures that are repetitive and time-consuming due to the repetitive nature of the operations themselves. Despite the fact that it is not the same as test automation, it may be used to complement traditional test automation, which will result in it being more adaptable and flexible.

The RPA testing method involves finding processes that are repetitive and prone to mistakes, and then automating those jobs. This process is known as "repetitive process automation." RPA allows developers to execute tests concurrently, which eliminates the need for them to manually test each and every feature of an application software individually, which might take several weeks to complete. They no longer have to rely on human validation as a result of this(Binder 2018).

Script less Test Automation:

There is a method of automating tests known as scriptless test automation, which does away with the need that human engineers be responsible for writing coded scripts. The term "no-code" or "codeless" test automation is another name for this kind of test automation. All of this is accomplished via the use of automation testing tools, which provide an abstraction from the code that is utilized for test automation. The writing and execution of automated tests is made easier for testers who are not technically aware as a result of this simplicity.

A significant number of individuals are of the opinion that scriptless test automation will be the method of choice in the testing process in the years to come. The key way by which the objective is to make testing more accessible, streamlined, and error-prone is to eliminate scripting, which is described as the use of programmed instructions that duplicate human behaviors. Scripting is the primary means by which the aim is to make testing more error-prone.

Shift-Left Testing:

The notion of "testing first-first" rather than "testing last" lies at the heart of the shift-left testing methodology. According to this definition, testers will have a greater role in the process of developing software from an early stage forward.

In this manner, testing is not an afterthought but rather an essential component of the methodology that is being developed. By moving to the left, development teams are able to identify problems early and handle them in a more effective manner, which eventually results in the saving of time and money while also ensuring that the software is of a high quality.

IoT Testing:

The number of Internet of Things (IoT) devices that are now in circulation is estimated to reach around 15.4 billion by the year 2023, as stated by Statista. The gadgets in question are actual computers that communicate wirelessly over the internet in order to collect and transmit data. When everything is said and done, it is expected that this number will ultimately rise. Not only that, but there is a reasonable explanation for this.

IoT devices have increased their capabilities, so people are now able to monitor and manage the health and status of their smart gadgets in real time, no matter where they are located. This is possible because of the enhanced capabilities of these devices. Smart refrigerators that read out recipes while you cook, smart sensors that monitor the operating temperature of industrial equipment, and smart GPS devices with geofencing to keep high-value assets within their designated zones are some examples of the sorts of devices that fall under this category.

As the number of software development teams continues to rise, more and more of them will need to include Internet of Things (IoT) testing into their SDLC. This is because the number of businesses that use Internet of Things technology into their products will definitely expand. Is that to be expected? The testing of the several components that make up an Internet of Things system, including the network layers, the operating system, the communication protocols, the hardware, and the software. In addition, it is vital to conduct security testing for the Internet of Things in order to prevent threat actors from gaining unauthorized access to test data. This is due to the fact that the majority of devices connected to the Internet of Things collect and transmit data that is significant to both individuals and businesses.(Tian 2005)

Cloud Testing:

The process of evaluating the functioning of an application program by using resources that are hosted in the cloud is referred to as cloud testing. Permitted testers from all over the globe are able to carry out a variety of testing activities, including performance, usability, integration, regression, and functional testing, via the internet in the cloud. This eliminates the requirement for on-premises infrastructure. Additionally, testers have the ability to expand their test coverage by altering their virtual environment in order to imitate a variety of desktop and mobile devices.

The use of this strategy eliminates the need to be concerned about the availability of hardware, and it is an excellent method for ensuring operation on a variety of devices and web browsers, both new and old simultaneously. Because cloud testing is quicker than testing on conventional on-premises infrastructure, it may also assist speed up the software development life cycle (SDLC). Additionally, cloud testing can help save money for both the customer and the development team by reducing the cost of ownership for testing tools. This arrangement is very beneficial to both of the persons involved.

Virtual and Augmented Reality Testing:

The global market for virtual reality (VR) is expected to grow to more than 22 billion USD by the year 2025, despite the fact that virtual and augmented reality are still considered to be relatively niche. The release of new devices, such as the Sony PlayStation VR2 and the Meta Quest 2, continues to push gaming forward by providing higher resolution displays, wider FOVs (Fields of View), and controllers that provide haptic feedback. Apple Vision Pro, which was only recently revealed and is scheduled to be released in 2024, is the company’s first effort into mixed reality. It gives customers the ability to manage the device using hand tracking and does not need controllers during operation.

There is reason to be optimistic about the future of the virtual and augmented reality industry, as these encouraging developments reveal. These findings also shed light on the growing significance of testing software using virtual and augmented reality in the software development process. Due to the extreme complexity of the two areas, it is necessary to conduct exhaustive testing on a variety of components. Among them are the user experience, the performance of the device, compatibility with a variety of hardware and software combinations, audio testing, and a great deal more. In addition, developers are required to do environmental testing in order to guarantee that the virtual experience is comparable to what a user would anticipate in the actual world.(Copeland 2004)

Automated Mobile Testing:

The term "mobile testing" refers to the use of mobile testing technologies for the purpose of automatically evaluating the functionality, reliability, security, and accessibility of a mobile application. Automated mobile testing may be accomplished with the help of certain technologies that are now accessible. These include LambdaTest, which allows users to do cross-platform testing by using Android and iOS emulators, and Appium, which is an open-source automation testing framework that enables users to write automation scripts in Python, Java, Perl, and other programming languages. Both of these tools are intended to facilitate testing across several platforms. These two options both provide the capability to test across a variety of platforms. If it is carried out correctly, automated mobile testing has the potential to deliver faster feedback and early problem discovery. Additionally, it has the capability of increasing test coverage to a bigger number of devices than was previously achievable with human testing.

API and Service Test Automation:

Since the introduction of what many people consider to be the first modern application programming interface (API) by Salesforce at the IDG Demo Conference on February 7, 2000, APIs have been extensively used by companies not just in the United States but also in other countries across the world. Businesses have been using these application programming interfaces (APIs) to connect third-party services and to monetize the content and functionality of their websites. However, despite the fact that application programming interfaces (APIs) have been around for quite some time, the testing process for them is always being refined. When it comes to evaluating the quality assurance of application programming interfaces (APIs) and making certain that they meet predefined requirements for performance, security, and compatibility, automation has become more common over the course of the years.

Karate DSL, which combines automated testing, mock-ups, and performance testing into a single platform, and SoapUI, which enables automated testing for REST and SOAP applications inside a DevOps framework, are two of the most popular automated API testing solutions. Karate DSL incorporates all three types of testing as well as performance testing. Karate DSL is also one of the most often used tools for evaluating application programming interfaces (APIs).

Big Data Testing:

Big data testing is the process of ensuring a big data application is processing its data properly.

Different from the testing of conventional software, which evaluates practically every component of a software program, the testing of big data involves analyzing the characteristics that are directly relevant to data collection and processing. This is in contrast to the testing of regular software. As a consequence of these checks, it is guaranteed that the data is not corrupted, and that it is also clean and well-organized. In addition to this, the tests uncover instances of data that has been misplaced or stolen.

What is the significance of testing using a large amount of data data? The reason for this is because it instills confidence in organizations, which enables them to trust their data and employ it to make choices that are based on correct information. When taking into consideration the fact that it is projected that the market for big data analytics will reach 68 billion USD by the year 2025, it is clear that the need for big data testing is only going to increase.(Cem Kaner, Jack Falk 2008)

DevSecOps:

When it comes to the development of software, the concept of making security a high priority throughout the whole process is not a new one. What is the reason why DevSecOps, which is an acronym that stands for development, security, and operations, is featured on this list? mainly due to the fact that a significant percentage of companies are still struggling with it. According to the findings of the study conducted on the 2021 State of DevSecOps, sixty percent of respondents acknowledged that they had encountered technical problems or difficulties with DevSecOps, while forty percent of respondents confessed to having encountered financial difficulties.

In the future, it is feasible that artificial intelligence and machine learning may be able to aid enterprises in attaining compliance with DevSecOps. The way is it? Automating the process of writing test scripts, delivering them, and auditing them is the means by which this objective is achieved. Additionally, the process of updating test scripts whenever there is a modification to the application need to be automated. This is something that should follow. Furthermore, artificial intelligence and machine learning may be of aid in continuous monitoring by discovering security vulnerabilities, flaws, and blunders at an early and frequent stage. This may be accomplished via the use of machine learning and AI.

QAOps:

The term "QAOps," which literally translates to "Quality Assurance and Operations," is used to describe the process of incorporating quality assurance into the DevOps methodology. Because of this link, it is now feasible for teams to simultaneously construct and test operating systems. The objective of Quality Assurance Operations, also known as QAOps, is to detect defects as soon as they are discovered. This is accomplished by conducting automated testing and locating issues before the products are delivered to the manufacturing line.

However, how does QAOps differ from other methods of quality assurance that are more historically accepted? When using QAOps, testing is not only carried out at the end of the development cycle, but it is also carried out constantly throughout the whole of the process of development and deployment. The speed at which feedback loops are closed is increased as a result of this, and the efficiency with which bugs are identified and fixed is improved.(Anne Mette Jonassen Hass 2016)

3. Literature review

3.1 Overview of Manual Testing

Software developers utilize a variety of test design testing methodologies to locate and fix errors. Software testing is a procedure used to make sure software acts as intended and only accomplishes what it was intended to do. Software ought to be dependable and unwavering, presenting users with no unexpected situations. A crucial step in the software development process is software testing. Even though testing is widely used and beneficial, the design phase is among the costliest in software development, carrying out, and maintaining the tests Software must be tested to guarantee a specific quality level, find errors, and confirm that it is correct. Software firms are always looking for methods to improve and maximize the efficacy and efficiency of their testing procedures. Writing and running tests by hand without the use of automation tools is referred to as "manual testing." Traditionally, software developers write these tests. High-quality test creation requires a lot of human labour, which in turn requires.

Big initiatives that require further testing may cause obstacles. Over the years, a few methods for automating the creation of test cases have been put forth, due in part to the difficulty and time-consuming nature of developing and specifying manual tests. Software organizations have already embraced the first step toward automation: automatic test execution. This usually implies that manual test writing is still required, but the tests are compiled into executable programs that are used to automatically test the system. A more sophisticated variation involves using test automation tools to automatically generate executable tests based on models or source code. The standard of the automatically produced tests differ, and it’s not apparent how these tests stack up against the ones that were created by hand. (Pai 2023)

When given enough time and resources, developers frequently provide high-quality tests; nevertheless, the focus of the tests varies depending on the software developer, the project’s goal, and the specifications that were developed by the community or company. Different developers and testers write different tests; some want to cover more ground, primarily by covering specific statements or branches;

others concentrate on strengthening code that is prone to errors or verifying that it complies with various standards. However, as software complexity and anticipated quality rise, so does the cost of developing the tests by hand.

Nevertheless, there are still not many sophisticated tools available for automated test generation, and as a result, there is little data comparing automated test generation to manual testing. A meta-analysis that organizes this data and offers a comprehensive comparison is required. (Ted Kurmaku 2020)

3.1.1 Types of Manual Testing

A wide variety of manual testing methodologies exist, each tailored to a certain set of needs and applications. Many of the most popular varieties are listed here:

3.1.2 Black box testing

The goal of this testing is to assess the software’s behavior and functioning without delving into its underlying workings or rules. The tester treats an application as a black box and demonstrates how it handles multiple inputs and delivers significant output. It is used to find out the behaviour& how the software works from the end user’s point of view. In order to understand how it works, let's take a look at a simple example.

For example, if you want to test a social application using black box testing, it will accept a username & password, and the expected result is access to the application multiple black box testing techniques validate the system against predefined requirements. Black box testing encompasses both functional and non-functional testing at various levels.

Functional Testing:

During functional testing, the quality engineer confirms whether the application components work based on specific requirements. This test can be done manually or with automated tools, depending on the specific test.

Non-Functional Testing:

Passive testing evaluates the functionality, reliability, usability, and other static characteristics of a software application.

Regression tests:

These software tests are performed after code updates to ensure that there are no bugs left in the updated code. New code may have new meanings that subsequently conflict with existing code and cause errors. This is why the QA team creates a series of regression test cases that will be repeated every time the code changes to save time and improve the testing efficiency. (Jorgensen 2002)

3.1.3 White box testing

Testing the architecture and internal code of software is known as white box testing or glass box testing. The default input is compared to the intended output in the test. To get this test, QA testers need to understand programming skills that focus on coding processes.

The primary goal of this testing is to strengthen the software's security by examining the software's output and production processes. Every line of code has been checked. After executing the white box methods, the developer submits the software to the testing team to conduct black box testing & validate the software with specific requirements. Here are examples of white box testing techniques:

Rules of coverage:

Code coverage is a test that helps understand how much testing is being done on sources. It is a useful metric that helps assess the quality of a testing program.

Road testing:

Method testing is a white box method based on system management. The program's settings provide the basis for testing.

Loop testing:

This is one of the key assumptions built into many algorithms. The purpose of this test is to reveal weaknesses in a particular loop. (Johnson 2019)

3.1.4 Gray box testing

This testing approach combines black box and white box techniques. Testing a system's functioning without understanding its fundamental setup is possible using the black box testing approach. The internal rule structure of a system is examined during white box testing.

Both integration and penetration tests often use gray box testing. As a whole, the system's components are tested during integration testing. Penetration testing involves conducting multiple scenarios that could lead to malicious attack attempts and identifying system vulnerabilities that resist such attacks. Testing techniques that can be combined with gray box testing include matrix testing, model testing, regression testing, and orthogonal array testing. (Sharma, A., & Gupta 2020)

3.1.5 Exploratory testing

A wide variety of manual testing methodologies exist, each tailored to a certain set of needs and applications. Many of the most popular varieties are listed here the term "exploratory testing" refers to software testing that is both unplanned and hands-on. System testing is a kind of testing in which testers analyze systems without looking at any test cases or having any prior experience with them. The personnel in question do not follow to a certain testing procedure; rather, they enter the room on their own and make a choice on the moment about what questions to test. The exploratory testing methodology is yet another well-known agile technique. The key goals of this methodology are to learn, discover, and explore.

In addition to the examiner's responsibility, it lays a focus on the examiner's autonomous nature. In circumstances in which the essential documentation is either not given at all or is only provided in part, testing may be of great assistance. The process of testing requires identification, which allows you to find a bigger number of flaws than you would be able to find using a standard test method. The quality assurance team is able to uncover errors and flaws that are often overlooked by traditional testing methods thanks to the use of this procedure. A variety of tests, as well as a variety of various scenarios and challenges, are included in the testing process. While the experiment is being carried out, it also generates new ideas from the process. (Bach, J., & Bolton 2010)

3.1.6 Usability Testing

Usability testing is used to evaluate the usability of software by asking users to perform and monitor specific tasks. Usability testing examines user behavior, specifically whether users can effectively, efficiently, and successfully use a service or software.

But UX experts use usability testing to explore experience beyond a practical understanding of usability. Usability testing is a process of collecting feedback and data about user experience that involves seeing testers engage with the product. Functional testing allows you to perform functional testing at any stage of the design or development process.

You can consider usability at the beginning, in design prototypes, later in the development process, in web pages or applications. It is recommended that functional testing be carried out during the development phase. Organizations need to focus on usability testing to save significant time, improve savings, and reduce costs. Thereby the organization ensures that the software release in the market delivers success. (Tullis, T. S., & Albert 2013)

3.1.7 Ad-hoc Testing

Ad hoc testing is a type of testing that is done on applications because it does not involve much preparation or planning. The purpose of post hoc testing is to identify bugs and problems at different stages of software development. Because it is not subject to any test protocols, there is no need for documentation or special procedures for developing test cases.

Ad hoc testing that can be performed at any time throughout the development of an application does not have a set schedule. However, it’s not as busy as it sounds. Testers need to have a good grasp of this programming skill. One advantage of ad hoc testing is that it takes less time than standard test methods.

You will save a lot of time because you didn’t follow a predetermined process and document each step. If you want to make sure your product is completely bug-free before it’s released to the public, ad hoc testing is a fantastic tool to use. Ad hoc experiments included the friend experiment, the monkey experiment, and the pair experiment. (Goyal, A., & Singhal 2018)

3.1.8 Regression Testing

It is a method of software testing that performs repeated functional and passive testing & ensures that the software apps perform at a high level through code modifications, software updates, enhancements, & optimizations. Regression testing is an important part of the software development cycle as it allows developers to detect unexpected errors in the app that are triggered due to growth & expansion of the codebase.

Testing validates the entire software/application by tracking components of existing functionality. It is an important step that should be implemented whenever an organization makes regulatory changes to its software. Testing ensures the stability of the system after repeated corrections. (Grigera, Pablo 2007)

3.1.9 Acceptance testing

The software’s acceptance is tested using this process. The goal of this testing is to determine whether the system is ready for release and if it complies with certain standards. Testing is done after the software is developed & before the release of the software available in production. Acceptance test is divided into two parts one is internal acceptance test & other is external acceptance test. Other types of acceptance tests include:

User Acceptance Testing (UAT): This testing is done by end-users & customer representatives to determine if the system meets the project requirements. This often includes a real-world scenario & ensures the software is ready to be used.

Business Acceptance Testing (BAT): The purpose of the following testing is to verify that the operating system and software meet user expectations and needs. In other words, a software product in BAT is tested as. (Kumar, S., & Sharma 2021)

3.1.10 Compatibility Testing

Compatibility testing determines whether a program or product is compatible with computer environments. It is part of the passive test. It evaluates the usability, reliability, and performance of the application and product. Browser and software compatibility testing is important because it allows organizations to develop applications that work well on virtually any device. For example, cross-browser compatibility testing ensures that Opera users get the same experience using Firefox and other major browsers.

Compatibility testing can identify potential problems when using an operating system or application in a particular environment. This allows you to resolve such issues before the system application starts, saving time and resources. Compatibility testing ensures that your application or system will work on different software and hardware platforms, which can help you expand your market share and appeal to a wider audience. (Bhatia, S., & Sharma 2018)

3.1.11 Performance testing

To ensure that software is secure, flexible, and functional, performance testing is an important part of the testing process. Consequently, testers often use a wide range of performance testing methodologies and instruments, tailored to the specific product under scrutiny. The system’s resilience to heavy loads, data, and peak demands is assessed during performance testing. This helps identify any technical issues or performance problems in the system.

Verifying that the system can handle the workloads expected by end users is the primary goal of performance testing. It helps to identify faults that may cause system failure in a particular situation. Additionally, workload testing helps to determine the capability of the system and its ability to handle different workloads. It is an important part of software development because it ensures accuracy and efficiency.

Performance testing guidelines are essential to ensure a smooth user experience, validate system reliability, prevent revenue loss, promote SEO rankings, and avoid future performance issues Performance testing is essential to ensure the smooth running of the systems and ensure a quality user experience. Identifying potential concerns before the development phase can also help avoid major challenges in the future. (Thakkar 2024)

3.1.12 Automated GUI Testing

When it comes to software, the system testing process includes GUI testing, which is sometimes referred to as GUI-based testing. So that we may learn more about the user-facing portion of the software, the tests are carried out. Inputs are accepted from users via graphical user interface (GUI) events such as mouse clicks, selections, and typing. These commands change the state of graphical user interface (GUI) elements like buttons, drop-down menus, and text fields, and then they produce graphical output, correspondingly. An outcome is produced by every event that may be performed on graphical user interfaces (GUIs), and this result is dependent on the GUI’s internal state as well as its external environment. Each event that is conducted on graphical user interfaces (GUIs) results in a distinct effect depending on the state. One last thing to consider is that the order in which an event occurs might also result in varied results. Graphical user interface (GUI) testing is difficult and time-consuming since every GUI event must be tested in different states and according to different sequences based on these characteristics. It is essential for testers to have a comprehensive grasp of the software's requirements and functioning to guarantee that GUI testing is carried out effectively. In addition to this, there must be a comprehensive comprehension of the many states that the program can exhibit as well as the conceivable sequences of graphical user interface events. A comprehensive test coverage of a system may be achieved with the use of this information. When testing software at the system level, graphical user interface testing is often a must. Its purpose is to guarantee that the program functions appropriately in accordance with its requirements. To do this challenging and intricate task, one must have an in-depth understanding of the software's requirements, capabilities, various states, possible results, and GUI event sequences. (Saha, D., Roy, C. K., & Kim 2020)

First Generation:

In their first generation, researcher include a reference to the initial level of GUI-based testing, which they refer to as the first generation. And so it is that this level is also called "the tolerance of changes in the GUI." A method of testing that involves creating changes to the graphical user interface (GUI) is called coordinate-based GUI-based testing. With this form of testing, the SUT's graphical user interface (GUI) is interacted with by means of precise screen coordinates. In its early iterations, Capture and Replay (C&R) applications could record user actions by using precise mouse positions to build and run test regimens. This allowed the systems to capture user activities. In addition to the fact that this method is no longer supported by the instruments that are now accessible on the market, it is also categorized as having the least amount of tolerance for changes. A simple change in the screen resolution would result in the test scripts that were written being corrupted, which a substantial would need amount of maintenance. This is the reason why this is the case. (Rashid, F., Mohamad, R. A., & AlSarayreh 2019)

Second Generation:

It is the components or widgets of the graphical user interface (GUI) that serve as the foundation for the second generation of automated GUI testing. This kind of testing represents an increased level of tolerance for changes in the GUI. A button, text box, or drop-down menu are all examples of capabilities that fall under the category of widgets in a graphical user interface (GUI). Widgets are defined as functionality that allows for user input. A widget’s background color, size, and font are a few examples of GUI components or attributes. Another example is the size of the widget. One advantage this level of abstraction has over the coordinate-based method is that it allows for the complete transformation of all recorded user interactions into widgets that make up the GUI. There are strategies that are built on APIs that make up this degree of abstraction. Because of this, the test scripts are more resistant to changes that are made to the graphical user interface (GUI), which in turn minimizes the costs that relate to updating the test scripts.

Third Generation:

VGT, which stands for visual graphical user interface testing, is the third generation of GUI testing. Tolerance for change in the GUI is at its maximum with this. To achieve the aims of identifying and interacting with the GUI, this specific solution employs picture recognition on screen grabs. Through the use of this particular kind of graphical user interface contact, which is also referred to as bitmap interaction, it is feasible to imitate customer behavior. It is possible to do this by supplying the SUT with automated instructions for the mouse and keyboard, seeing the output, and then comparing it to the results that were predicted. Because of this, VGT is more resistant to changes in layout when compared to the second generation. It does, however, make the graphical user interface (GUI) more important, especially for picture manipulation (e.g., resizing, cropping, and color correction). Attention in the parts that follow will be focused on the horizontal axis of Figure, which indicates the level of automation in the GUI testing procedures. Separating script-based GUI testing from script-less GUI testing is an important first step toward this objective. (Nguyen, T. T., & Kapfhammer 2021)

3.1.13 Generations of Automated GUI Testing

Automated graphical user interface testing has been around since the late 1980s and continues from that point forward. A categorization system has been devised by researchers for the many different computer-based graphical user interface testing approaches. The currently available methods for evaluating graphical user interfaces are separated into three generations by this categorization scheme, which are temporally connected to one another. Figure shows the researchers’ more thorough classification of the current GUI testing approaches; the vertical axis shows the GUI/SUT tolerance for change, which is divided into four categories and is also called GUI-based testing generation.

Despite being the fourth highest level described by the authors, "combining visual and widget-based approaches" is considered the 3.5th generation as it does not introduce any new approach to the area. In addition, the authors included this level. The degree to which regression testing is automated via the usage of the graphical user interface is shown along the horizontal axis. These testing methodologies are investigated in more depth and explored in greater detail in the subsequent sections of the literature study that are to follow. (Aals 2019)

By using script-based graphical user interface testing, Section 3.1.12 has provided a taxonomy of automated GUI testing procedures. This taxonomy of testing approaches has been supplied. A visual representation of the recommended classification system that is being used by is shown in the figure. There was some back-and-forth over the meaning of the vertical axis of this graph, which shows the level of tolerance for GUI tweaks. Additionally, this degree of tolerance is also known as generations of automated graphical user interface testing. In this part of the literature study, we take a closer look at the graph’s horizontal axis, which shows the level of automation in regression testing performed via the GUI. Annotated in the issue description is the fact that this thesis differentiates between script-based and script-less forms of GUI testing.

This part begins with the script-based processes that are stated by, and it continues with the investigation of various GUI and script-based tests. One form of automated testing technique, script-based testing may create test scripts in real-time while test cases are running. This type of testing approach is often used in software development. Keep in mind that a tool that doesn’t need any code at all is different from a script-less tool. This is a very important distinction to get to. There is a chance that a tool that does not need any code might also be script-based. This is because the tool could create and run test scripts in a covert manner. A more comprehensive analysis of the script-based GUI testing methods currently available in the literature is presented in the sections that follow. (Nguyen, T.T., Memon 2017)

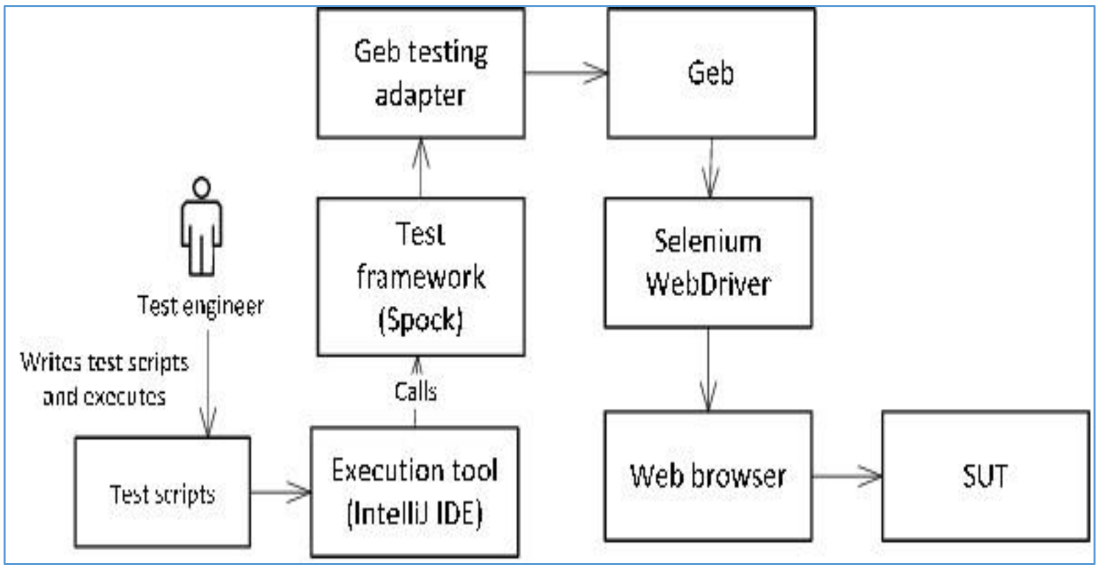


Figure 1: Automated GUI Testing Architecture.
[**Source:** This thesis specific diagram was developed by the author.]

Capture & Replay:

Figure 1 illustrates that the first degree of automation is referred to as Capture & Replay (C&R), which is also referred to as Record & Play to certain people. When it comes to automated regression testing, this technique is often regarded as being among the first and most widely used approaches out there. As far as C&R techniques are concerned, the instrument does exactly what its name suggests: it is a recorder that keeps account of all the inputs, which are then carried out by a manual tester. These inputs are saved as a test script, and they include a variety of graphical user interface (GUI) events. Some examples of these inputs include a mouse click or a textual input. For carrying out the automatic execution of the script, all that is necessary is to play back the test script. These technologies provide two separate modes: the capture mode and the replay mode. Both the modes include video capture. (Garousi, V., Khezrian, M., Felderer 2018)

During the time that the tool is in capture mode, it will record and preserve any kind of input that is supplied by the tester. In order to determine the item that is being evaluated, this input may be an event as well as the characteristics of an object, such as its color, position, name, and so on. This is done in order to accurately identify the item. In addition to the fact that the test script that includes all of these gathered attributes is retained, the replay mode allows for the script to be replayed several times. Within the context of C&R approaches, it is feasible to include the anticipated outcome of a use case as checkpoints.

These checkpoints are responsible for carrying out a comparison between the anticipated outcome and the actual output from the use case. To determine whether or not the SUT is behaving in an acceptable manner, it performs this action. This happens either while the capture mode is engaged or when the test script is being modified. Both scenarios are possible. In the event if a checkpoint in the test script discovers that the actual result is not the same as the one that was planned, the test is regarded as having failed. The behavior of the graphical user interface (GUI) may be recorded by tools that are more sophisticated and up to date. Therefore, these tools can identify modifications that have been made to the GUI in later versions.

The C&R methods are often the most convenient for regression testing since they are simple to use, need less human intervention, and provide data more quickly. The problem with these systems, on the other hand, is that they do not allow for alterations to be made to the graphical user interface (GUI). Both the test script that was recorded from earlier versions and the script that must be re-captured for the new graphical user interface need to be kept. Both test scripts were recorded from earlier versions. To maintain these scripts, there is an additional amount of human effort that is that is necessary. Ranorex, QF-Test (QFS, 2023), and Squish (Squish, 2023) are just a few examples of the many C&R tools that are now accessible. These tools are available in both open source and commercial versions. In addition to enabling C&R testing, these technologies also provide code-based testing, which will be covered in further depth in the next portion of this thesis. (Huang, Y., Huang, G., Leung 2019)

Model-based Testing:

Model-based testing, which is also often referred to as MBT, is yet another way that is used very regularly. This technique is separate from the ways that were mentioned before since it automatically creates the test scripts. It is vital to note that this strategy is distinct from the approaches. Additionally, it is conceivable for codeless automation tools to generate test scripts in the background, in a manner that is analogous to how MBT tools do their tasks. The fact that this is the case suggests that not all script-free technologies are also code-free automation solutions at the same time. To get a more profound understanding of the idea, it is necessary to place an emphasis on all the many implications that are associated with MBT. “The phrase "model-based testing" refers to a technique that, depending on the tests that are generated by the technology that is being used, might be interpreted in a number of different ways. Four various approaches are included into it, and these approaches may be classified into the following categories:

- Generation of test input data from a domain model.
- Generation of test cases from an environment model.
- Generation of test cases with oracles from a behavior model.
- Generation of test scripts from abstract tests

The implementation of the first strategy places a main emphasis on the automated synthesis of test inputs as its primary objective. To produce test input data, this strategy involves selecting and combining a subset of the values that are provided. This enables the generation of test input data. To focus on the development of test inputs is the objective of the second strategy, which aims to accomplish this. It is important to develop a model that contains data on the domains of the available input values in order to carry out model-based testing. This is a prerequisite for the testing process. This is done with the purpose of producing test input, which is also referred to as test input generation in a more casual sense. Take into consideration that this technique does not provide any information for test oracles, and consequently, it does not generate a result for the test. This is something that you should keep in mind. Keep in mind that this is a very important topic. (Arbon, J., & Burnett 2012)

This model is built as part of the second approach of MBT, which entails the building of a model that offers a description of the environment that the SUT is predicted to be contained inside. The next step is to utilize this model to generate predictions about the environment that really exists. Since these environment models are now available, it is now feasible to construct sequences of calls that are aimed at the SUT. The SUT is the component that is accountable for the generation of call sequences of this kind. It is not possible to determine whether the test is successful in a manner that is analogous to the approach that was taken before it. This method is like the one that was done before it. It is difficult to predict the values of the output since the environment model does not include any information about the behavior of the SUT. This is the reason why it is impossible to predict the output values. This is the reason why things are the way they are.

The development of test cases via the use of oracles is the third way that is utilized in MBT process. The output values that the SUT is expected to accomplish are indicated by these oracles, which offer an indication of those values. This model is referred to as the behavior model, and it is the name that is given to the model after it has been completed. Oracles that define the behavior of the SUT, which is the relation between the values that are input and those that are output, are included in the test cases that are created. This is the reason why this is the case. This is the reason why things are the way they are. However, in contrast to the other ways, it not only produces the input but also provides the tester with entire test cases and a test result. This differentiates it from the other approaches. An important benefit is that this is the case. Comparatively speaking, this stands in stark contrast to the several other testing methods.

A low-level test script that may be performed is at the heart of the last approach, which is based on the abstraction of a description of a test case. Examples of this kind of description include a sequence diagram or a uniform modeling language (UML). The model that is presently being created includes a description of both the structure of the SUT as well as the application programming interface (API). Tricentis, which is an example of a test automation tool that is based on the model-based approach, provides a detailed description of Tosca, which can be retrieved by the user.” It is possible to read a comprehensive explanation here. The third strategy of MBT is the method that is relevant to this research since it applies to the situation. This is because the major focus of this thesis is on the creation of test cases as well as the overall test design of automated testing. Model-based test cases may be developed using one of two unique approaches. Both of these approaches are described here. Each of these approaches is categorized in accordance with the category that was proposed by the investigating researcher. A description of the process of creating test cases that is based on models that were developed manually and those that were inferred automatically using the models is provided in the following paragraphs. (Bell 2018).

3.2 Overview of Automation Testing

The testing and quality assurance of software-intensive systems accounts for around 26% of IT expenditures, according to a 2014 industry poll of 1,543 executives from 25 countries. However, the cost of not testing is significantly higher. According to 2013 Cambridge University research, finding and fixing software vulnerabilities now costs $312 billion worldwide. It accounts for half of the typical project’s development time per year. Both automated and manual testing are important parts of the testing process. By simulating real-world user actions, human testers do manual testing to ensure that software under test (SUT) features work as expected. A documented test plan with a series of test cases is frequently followed by the tester to guarantee testing is thorough. The automation of software testing procedures is known as automated software testing. To be more specific, test automation refers to the process of independently testing software by use of specialized software that controls test execution and compares actual results with projected outcomes. The relative merits of automated and human testing are dependent on a variety of factors. When test automation is used, the initial inclination is typically to use it for tasks that human testers traditionally performed by hand. Still, (REHKOPF 2022)

While automation may reduce labor costs, it cannot fully replace human testers. The success of automated testing depends on the proper and appropriate implementation of test automation.

Test automation has the potential to significantly reduce testing costs and improve software quality when used appropriately. A recent poll titled the "World Quality Report 2014-2015" by the French business Sogeti found that just 28% of test cases are now automated, despite managers’ hopes for an improvement in the future.

The decisions on what and when to automate become increasingly crucial as test automation spreads throughout the software business. Making the wrong choices in this regard might result in disappointments and significant mistakes in expenditures.

While many individuals envision fully automated testing in an ideal future, a 2012 poll found that only 6% of practitioners agreed with this viewpoint. It is confirmed by further sources like and our surveys of test procedures in Turkey and Canada that practitioners believe that time and money restrictions prevent them from automating all tests. One frequently requested question is "what are the best times to test certain parts of the system automatically?". As of this writing (March 2016), a Google search for the subject "when to automate" software testing returns over 17,500 hits. Among these results is a plethora of online forums, conversations, and the sharing of personal experiences.

Additionally, a practitioner-oriented book titled "Just enough software test automation" emphasizes the significance of the subject and illustrates that automating software testing is not necessarily a yes-or-no decision. Some additional sites make similar assertions, such as: "Just as with any other testing activity, a cost-benefit analysis is used to choose which tests to automate. If the analysis is wrong, you’ll end up distributing your sources incorrectly. The formal literature (such as journal articles and conference proceedings) and the informal literature (such as online discussion forums and white papers) have both investigated the relative merits of automated and manual testing in regard to various parameters. (Binder 2013)

The topics of when and what to automate have been extensively studied by academics and industry professionals in technical publications, blogs, and forums. However, to until, there has been a dearth of secondary sources—i.e., "review" or survey papers—that analyze, gather, and synthesize the available information about the "what" and "when" of automated testing. Our goal in doing this Multivocal Literature Review (MLR) was to bridge the gap between theoretical research and the practical information that is required by businesses. In addition to books and academic studies, this investigation drew on a vast array of gray (unpublished nonresearched) internet resources, such as presentation videos, tools, blog entries, and white papers. One subset of SLRs, MLRs draw on both official and informal sources to compile their findings. (Vahid Garousi 2016)

3.3 Benefits and Drawbacks of Each Testing Approach

An Organized Review of the Literature Bruno Rossi, TomáŇsPitner, and Katar´ınaHrabovsk´a are faculty members at the Masaryk University in Brno, Czech Republic. Abstract: Software testing is crucial to raising the calibre of products. Several software testing methods have been created to assist enterprises as a result. Nevertheless, there is still a lack of consistency in the implementation of testing process models throughout enterprises; additional data regarding reported experiences is required. Objective: To ascertain the outcomes obtained from the utilization of software testing methods in organizational settings. Our attention is directed towards attributes like the testing procedure phases, stated benefits and downsides, and the context of use. Method: We conducted a Systematic Literature Review (SLR) that was informed by findings from earlier reviews and concentrated on research concerning the utilization of software testing procedures. Results: We gathered 17 testing models from 35 primary studies and survey-based articles. While many models now in use are said to be relevant to a wide range of situations, research findings indicate that certain models are insufficient for specific domains and not appropriate for all sizes of enterprises. In conclusion, the SLR evidence can be used to assess the applicability of various software testing approaches inside businesses. Benefits and downsides, as documented in the examples studied, help to clarify the advantages and disadvantages of each approach. A crucial step in the software development process is software testing. This review's objective was to present an overview of current software models that can be used to enhance the testing procedure. Utilizing the Systematic Literature Review (SLR) technique, we sourced empirical research that detailed the advantages and disadvantages of applying the models. During the process, a total of 17 testing models were identified. Their areas of application and model representation are different. Although most of the models are said to be broadly applicable, several strategies were modified to meet particular specifications for fields including military systems, automated testing, and embedded software. We also paid attention to the areas that the concrete examples' testing models helped to enhance. The majority of empirical research highlight process standardization as the primary improvement, after the model's adaptation to the testing procedure. The enhancement of product quality is the next crucially supported feature, and numerous research concentrate on certain elements like measurements or the identification and/or mitigation of flaws. Under some conditions, the benefits and downsides of a testing methodology may have an impact on its acceptance. Aside from the area or various improvement processes, models offer a number of special benefits. Based on the data gathered, we discovered that while most of the models currently in use are deemed generally applicable, some organizations believe the models are inadequate for use in specific domains like embedded software development or small-to medium-sized businesses. Additionally, the models use various procedures throughout the phases, which can be important when choosing a model. (Katar´ına Hrabovsk´a, Bruno Rossi 2019)

3.4 Automation Testing Tools and Frameworks

Software testing must be completed more quickly and successfully in order to guarantee that the standard is met, as there is an increasing need to offer high-quality software "Quality at Speed." A software testing project can only be successful and effective if it makes use of the proper test method(s) and test automation tools/framework. It is frequently necessary to combine a few suitable testing methodologies to thoroughly test software and guarantee that it meets standards. Similarly, no single tool can fulfil all automated testing requirements, which complicates the process of identifying the ideal tool combination. To achieve a successful and effective software testing process, the first step is to be aware of the many testing methods, tools, and frameworks. An extensive analysis of frameworks and technologies for test automation is presented in this article. First, the categories of automated testing were discussed, and then the different test automation frameworks were explained. Lastly, a succinct comparison and explanation of some of the most popular automation technologies was given. These days, good software testing requires the use of test automation. The latest World Quality Report 2018-2019 states that the biggest challenge to achieving "Quality at Speed" is test automation, which has many benefits such as saving time, reducing expenses, increasing efficiency, and improving accuracy. Therefore, without the appropriate automation tools and framework, successful and effective test automation cannot be accomplished. This report offers a thorough explanation of the various automation tools and frameworks as well as insights into some crucial considerations for automation tool and framework selection. (Mubarak Albarka Umar 2019)

3.5 Challenges and Limitations in Automation Testing

The opinions of academics and practitioners regarding software testing are known to differ. To bridge the divide, this paper examines both perspectives on the advantages and constraints of test automation. A comprehensive examination of the literature is used to examine the academic viewpoints, and a survey containing responses from 115 software professionals is used to evaluate the practitioner viewpoints. Based on a thorough literature assessment, just 25 studies give sufficient evidence about the benefits and limits, indicating a shallow supply of information. Additionally, it was shown that restrictions frequently came from experience reports, whereas benefits frequently came from stronger sources of data (experiments and case studies). We think that the publication bias of favorable findings is to blame for this. According to the poll, test automation offers advantages in terms of test coverage, reusability, and repeatability as well as reduced work required to execute tests. The constraints were large upfront costs for setting up automation, choosing tools, and providing training. Furthermore, 45% of the participants concurred that the current tools on the market do not adequately meet their requirements. Ultimately, the belief that automated testing will completely replace manual testing was rejected by 80% of practitioners. Three contributions are made by this paper. Initially, we conducted a thorough analysis of the advantages and drawbacks of software test automation in scholarly works. After filtering out 24.706 papers, we had 25 research works (see Table II). Consequently, the body of evidence supporting these claims is somewhat thin because only one or two sources support many of the restrictions and benefits. Additionally, we discovered that whereas the strongest sources of evidence (experiments and case studies) typically yielded benefits, experience reports were more likely to disclose limitations. We believe that publication bias over the benefits is to blame for this. We think that doing thorough empirical investigations, such as case studies and experiments, to evaluate the constraints of test automation is crucial to future research in this field. (Rafi 2021)

3.6 Case Studies on Manual and Automated Testing Implementation

One recommendation for automated test generation is a less expensive method of producing tests. However, the comparative analysis between these exams and hand created.

Those that are cost-effective and convenient. Industrial control software in particular often needs extensive manual testing to fulfill stringent specifications for both code coverage and testing based on specifications. For this reason, we conducted a case study comparing the results of tests generated automatically with those generated manually. We used fresh, real-world industrial code developed in the popular programming language IEC 61131-3 for building industrial control systems based on programmable logic controllers. On average, automated tests boost code coverage by 90% in a fraction of the time it takes human testers to do the same task. Comparing automated test generating tools to manual testing, we discovered that the former did not improve defect identification in terms of mutation score. In particular, mistakes of the logical, timer, and negation types are better caught by human tests than by automatically produced ones. Further research on the efficacy of automated test generation in the creation of trustworthy systems and the methods utilized for manual testing in industrial settings is required considering these findings. The purpose of this study was to compare the efficiency and effectiveness of manual testing with automated test creation in a business environment. A newly developed industrial control software and 61 IEC 61131-3 programs were used in the research, which relies on test suites that were manually constructed by industry specialists.

Our research shows that automated test generation can provide decision coverage on par with manual testing by industrial engineers, but at a much lower cost and with far less effort. While these computer-generated test suites may not provide better fault diagnosis in terms of mutation score compared to human written test suites, they do have one advantage. Our results are consistent with those of other research that has shown a similar problem identification rate when comparing manual testing with automated code coverage-based test development. Surprisingly, our findings suggest that manual test suites may be marginally more effective at detecting errors than comprehensive test-based test suites. To prove this idea statistically, however, bigger empirical research is required. Our research shows that compared to machine produced test suites, human designed test suites are more likely to catch certain types of errors. Automated test creation for industrial control software might benefit from a more targeted mutation testing approach if it were to provide test suites capable of identifying certain defect kinds. (Nguyen, H., & Tran 2014)

3.7 Best Practices in Combining Manual and Automated Testing

An essential step in ensuring the calibre and dependability of software products is software testing. Software testing is commonly approached through two methods: automated and manual. While automated testing uses software tools to run tests automatically, manual testing involves testers carrying out test cases by hand. Choosing between human and automated testing has a significant influence on the efficiency and efficacy of software testing. Within the context of both human and automated testing, this research paper analyses the efficacy and efficiency of software testing. Data from a software testing project that combined automation and human testing methods is analysed in this study. In order to assess the efficacy and efficiency of human and automated testing in terms of duration, expense, and test coverage, the study used quantitative analysis. According to the study’s findings, automated testing is more cost and time-effective than manual testing. In terms of test coverage and problem discovery, however, manual testing performs better than automated testing. The report also points out the benefits and drawbacks of each strategy and recommends a hybrid strategy that includes the best features of both automatic and manual testing. This study advances knowledge regarding the effects of automated and manual testing on the efficacy and efficiency of software testing. Based on the particular requirements of the software project, this study offers software development teams insights to help them decide which testing strategy to use. This study aimed to investigate the effects of both automatic and manual testing on the efficacy and efficiency of software testing. Data from a software testing project that included automatic and human testing methods was examined in the study. The efficiency and efficacy of human and automated testing in terms of time, cost, test coverage, and defect identification were compared using quantitative analysis in the study. The study's conclusions showed that, in terms of both cost and time, automated testing is more effective than manual testing. But manual testing is more efficient in terms of test coverage and defect discovery than autonomous testing. Additionally, the study highlighted the advantages and disadvantages of each strategy and recommended a hybrid approach that incorporates the best aspects of both automatic and manual testing. The study's conclusions have significant ramifications for teams who develop software. The particular needs of the software project should guide the decision between manual and automated testing. Testing efficiency can be raised, testing time and expense can be decreased, and testing can be done automatically. To guarantee sufficient test coverage and to find flaws and problems that automated testing overlook, manual testing is still required. The study concludes by highlighting the significance of selecting the proper testing strategy in accordance with the needs. (Khin Shin Thant 2023)

3.8 Trends in Software Testing

The Software Development Lifecycle's testing phase is the most important since it determines if the product will be delivered in its finished form. There has to be innovation and improvement in this process since it is labor-intensive and time-consuming. (Garg 2018)

Because of this, it's important to use automated testing and a variety of test metrics before and throughout testing. Time savings and the creation of a reliable, effective, and efficient final product that not only meets but exceeds all criteria are both possible outcomes of using this approach to testing. The platform that houses software testing and development is still very good and is always changing. But something as important and vital as testing is frequently added very late in the software development process. For improved comprehension and early review, which may resolve ambiguity issues and thus save the cost of later software fixing, there should be a maximum amount of contact between specification writers and testers. Once testers are aware of the requirements and standards, they should provide developers with a specific lightweight test model so they may create as soon as the project is handled for official testing, make sure the primary specifications are met. Testers can greatly benefit from using simulation tools to create an environment that is comparable to the one in which the product will function. This allows for the best determination of exception handling techniques and specific exception testing. The product may be tested in an environment that closely resembles its intended use by simply integrating the simulation into the testing procedure. Consequently, testing-related labor in the future will rely heavily on technology, with automated testing models and simulation helping to shorten the testing life cycle, provides optimum problem avoidance, and provides effective quality assurance.

Test case generation based on manually created models: According to the researchers suggested classification system, there are two distinct groups for MBT as well. First, there's MBT, which relies on models that have been hand-crafted in order to generate test cases. The test designer is tasked with modeling the user interface and its intended behavior in order to use test automation techniques. The test cases are automatically generated by the tool using the behavior model that was manually built. There are a number of methods that can be explained, but all of them involve manually building models and then automatically creating test cases from those models. These methods include keyword-driven models, faulty event sequence graphs, AI planning, genetic models, probabilistic event-flow graphs, latin squares, coverage arrays, hierarchical finite state machines, and a tactic based on UML diagrams. (Talha Ahmed Khan 2021)

Test case generation based on automatically inferred models: Model ripping, model extraction, or model inference are some of the more contemporary terms for the process of automatically extracting GUI models that were introduced in MBT. Traditional model extraction methods relied on static examination of the system's source code, which made it impossible to account for the GUI's dynamic behavior. Hence, similar to C&R tools, dynamic analysis was implemented to examine the GUI's behavior while the user interacted with the SUT. Automatic manipulation of GUI widgets is now feasible thanks to the interaction's ability to mimic human input. A number of methods exist for automatically extracting models, such as the feedback-based model extraction methodology, event interaction graphs, and event flow graphs.

Script-less GUI testing:

Techniques that do not rely on scripts to generate test cases are known as script-less testing methodologies. Which means they are script-less, as no scripts are produced to run the test cases. By picking and performing the actions of the found GUI states, a script-less method dynamically produces user actions sequences during runtime to explore the SUT. Following these steps will allow you to use a script-less tool with ease. The initial step for script-less methods and tools is to determine whether SUT GUI widgets are now accessible. Then, from those GUI widgets, we deduce all the available actions. Using an Action Selection Mechanism (ASM), choose a subset of these activities to construct the test sequences is the next step. The random testing technique is being used to do this ASM. Further discussion of this method follows in the section that follows.

Random or Monkey Testing:

Software testing methods that do not include scripts are called random testing, monkey testing, or stochastic testing. In this method, test cases are produced in a completely random manner while the test is being executed. In this approach, the SUT is probed by means of generating random inputs and performing random actions in order to identify broken systems. Due to the fact that this technique is primarily concerned with sequences that identify failures in the SUT, the majority of the time, test cases that are produced during testing are not stored. This strategy has the benefit of not requiring the construction and maintenance of test cases, which is a significant advantage. Monkey testing, in contrast to script-based procedures, has the ability to uncover bugs that scripted approaches are unable to uncover. Microsoft said that test monkeys are responsible for discovering between 10 and 20 percent of all issues. When referring to test automation tools that are based on the random testing technique, we use the phrase "test monkeys" to characterize these products. The reason these test monkeys may see the SUT from a different angle than a human tester is because they often explore the SUT in a different way while doing the tests. Making the SUT unresponsive and crash is a common goal of test monkeys, therefore they construct random test sequences with that intention. One of the tools that is based on the monkey testing approach, which can be found in for a more in-depth description. Test monkeys may be divided into two categories: clever monkeys and stupid monkeys. Both of these categories are used in the procedure of monkey testing. (Anil Kumar, Anuj Kumar 2017)

Smart Monkeys:

Due to the fact that they possess some information of the SUT, monkeys are referred to be "smart" monkeys. They are aware of the action sequences that may be used to perform a straightforward task and the manner in which that functionality should be executed. Capabilities that fail to deliver the desired outcome are considered failures. Making the monkeys acquire information about the SUT is the first stage in the process of developing clever monkeys. By using an application programming interface (API) or image recognition to identify the widgets shown on the GUI, one may retrieve the programmatic structure of the GUI's layout and widgets. During the second step, the information that was gathered during the first phase is used to identify and extract the current state of the graphical user interface (GUI). Once the monkey has gained some familiarity with the graphical user interface (GUI), it will be able to deduce a series of actions that will be carried out.

To crash or jam the graphical user interface (GUI), the objective of clever monkeys is to produce arbitrary input sequences. The information about the SUT may be gathered via the use of a state model, which also assists with action selection and directs the clever monkey through the graphical user interface (GUI). It is the action selection mechanism (ASM) that is responsible for this. In script-less testing, one of the most important steps is to choose the appropriate actions to perform, since the detection of errors is dependent on the activities that are being carried out. Action selection may be carried out in a completely random fashion, or it can be accompanied with some kind of intelligence in order to provide the test monkey with instructions on what to do next. It is possible to increase the intelligence of an already intelligent monkey by using techniques such as meta-heuristics, reinforcement learning, and ant colony optimization (ACO). During a particular state of a graphical user interface (GUI), the purpose of each of these methods is to determine the most effective course of action for a test monkey to take.

Dumb Monkeys:

On the other hand, dumb monkeys conduct themselves in a way that displays their lack of comprehension throughout the testing process. They are entirely unaware of the SUT and act in a manner that reflects their lack of knowledge. They are unable to identify the present state of the SUT or the results that may be predicted following the execution of an activity since they do not own a state model. This has prevented them from being able to determine either of these things. During their interactions with insects, they demonstrate a lack of awareness, which ultimately results in an unexpected event that the dumb monkey is unable to recognize. Among the things that they are able to recognize are, for example, flaws that are easily observable, such as crashes and hangs. In the late 1980s, Apple built the first dumb monkey test tool with the intention of analyzing the resilience of their software within the environment of their operating system. This was done in order to determine how well the program would perform. It is common practice to use dumb monkeys for the purpose of detecting flaws in operating systems; nevertheless, these monkeys are also capable of locating errors in programs. These applications are not only economical but also easy to develop, which makes them more desirable to experts who test software. This is because they are completely automated, which makes them simple to design. (Kumar 2019)

Challenges of GUI Testing:

Since GUI testing is an emerging topic of software testing, we've already established that it's no easy feat. We discussed the key issues highlighted in a recent literature study that outlined the difficulties encountered by researchers in the area of GUI testing. This section will discuss the problems that have already been identified in the literature and will provide a summary of the drawbacks of the testing methods that have already been stated. Automated GUI testing is addressed in the first part, which also discusses the more general difficulties of test automation. The difficulties with the aforementioned automated GUI testing methods are going to be discussed in greater detail in the parts that follow. The most common problems with automated testing Due to continuous integration procedures, short development cycles of incremental and iterative processes, and other issues, system-level testing via the GUI is a big difficulty in software development. Testing must also occur in incremental phases due to iterative and incremental development, which is a byproduct of shorter development cycles seen in agile approaches. Maintenance of test cases and scripts, as well as regression testing, are all part of quality assurance operations that are significantly cut down by this. It is possible to reduce testing quality and inefficiency due to a lack of time for regression testing.

In addition, the development cycle’s testing operations are becoming more costly and time-consuming due to the human execution of regression tests due to a lack of time to automate test cases. Testing efforts are also rising in tandem with the complexity of software systems that support several devices and platforms, since each test run must be executed on all of these platforms and devices. Some examples of this kind of testing include compatibility testing, which checks whether the SUT can work with various versions or facilities of software and hardware, and cross-browser testing, which involves testing via several browsers that the SUT supports.

Among the many difficulties with test automation is the test oracle issue, which presents even another obstacle. One definition of a test oracle is a function or activity that checks whether a certain sequence of operations is acceptable for the SUT. Because mathematical logic defines specified test oracles, a specification language is necessary. An SUT’s proper and improper behavior may be identified by a derived test oracle using artifacts such as documentation, system executions, and versions of the system. Finally, a formal specification and domain expertise are not necessary for an implicit test oracle to differentiate between an SUT’s proper and improper behavior; instead, it depends on implicit knowledge. The tester who determines whether the SUT’s behavior is right or wrong is known as the human oracle.

The primary obstacle to increasing test automation in software testing is the paucity of focus and study on automating test oracles. Reviewing and checking the behavior of SUTs still requires human interaction. One drawback of the previously stated test oracle techniques is that they do not have a formal specification. As a result, they depend on the abstraction of models, which might lead to imprecision or include unnecessary behavior from models like the GUI model. Interpreting the abstract outputs is another difficulty with these methods; this makes it hard to comprehend the test findings and necessitates human involvement. The need to manually define oracles is a hurdle for automated testing methods. Since it is not possible to describe anticipated behavior for every SUT state, manually specified test oracles could lead to test results that are unclear. (Gaikwad. 2020)

Challenges of script-based GUI testing:

The next part will go into more detail on the difficulties of script-based testing and every testing method within that domain. The following are some of the potential problems that may arise while using various testing methodologies, in addition to the test oracle problem.

Capture & Replay:

The high maintenance costs and effort required for test scripts are a downside of capture and replay systems. This method gives the least amount of automation in regression testing and has the lowest tolerance for changes to the GUI, as illustrated in Figure 1 and detailed in Section 3.1.12. As a result, this method requires a lot of upkeep and a lot of human labor to modify test scripts, particularly for regression testing. This is why C&R tools are often reserved for first software releases—to facilitate quicker feedback—and subsequently retired after the introduction of new SUT versions. Adding additional test cases to an existing test suite is another potential stumbling block for testers using C&R tools. Adding new test cases to existing test suites may need programming, which in turn may necessitate more programming knowledge and time spent manually. This difficulty highlights the fact that new versions of SUTs might result in longer test runs and greater expenses, especially when C&R tools are used for rapid feedback on early versions of SUTs.

Model-based Testing:

Model-based testing is gaining popularity and is being integrated into several test automation systems. However, how well these tools enable model-based testing greatly influences their practicality. Since test scripts need to be constantly updated for regression testing, MBT methodologies, similar to other script-based testing techniques, are also not resistant to GUI changes. Tools for automating tests that use the method of manually constructed GUI models (discussed in Section 3.1.12) still need human intervention and specialist knowledge in order to construct the models. In most cases, in-depth training on these tools is necessary for the tester to acquire the necessary expertise. It may be a huge pain to figure out how to utilize the tool and make the GUI models to automatically produce test cases; it also takes a lot of time and money. (Saba Khalid 2021)

However, the problem with automated model extraction approaches is that they still need human input. For instance, a tester may need to manually predefine certain inputs, such usernames and passwords. The fact that the automatically produced model still requires human inspection and correction to achieve the target coverage of GUI models demonstrates that, despite the reduction in manual work, human involvement is still necessary. Another issue is that testers are only concerned in the anticipated and needed behavior of the SUT, but models' irrelevant behavior is also being extracted. Another obstacle that has to be addressed in the sphere of MBT techniques is validating the validity of such models.

Challenges of conducting GUI testing without scripts include an approach known as random or monkey testing, which involves choosing actions randomly. The subsequent section explores the disadvantages of this method identified in the literature, as well as the test oracle problem highlighted in the previous part.

Monkey Testing:

The very lengthy execution time of the monkey testing technique is a significant concern. Since it seeks the vulnerability that causes the SUT to crash, a test automation tool based on this method may have test runs that persist for several days (Aho&Vos, 2018). To get a high level of testing coverage, monkey testing requires a high execution time. Improving the resources and doing parallel executions is one way to get over this problem, but it won't make the test run as quickly as the other methods. The monkey testing method was the subject of several published studies that detailed its shortcomings when used to system-level testing. When compared to other testing methods, these academics consider monkey testing to be the least effective and are not confident that it can identify serious errors.

The replication of the found flaws is a big problem with monkey testing after the lengthy execution time. A test monkey may need many days to crash an SUT, leading to testing sequences that are both lengthy and arbitrarily generated by arbitrary GUI actions. For debugging reasons, it might be rather difficult to reproduce certain errors, making it much more difficult for the developer to remedy the detected fault.

In monkey experiments, intelligent and non-intelligent monkeys are distinguished. The SUT and its condition are completely unknown to ignorant monkeys. It is completely unclear to them what is and is not acceptable input and output. As a result, they miss the mark every time they encounter a glitch. However, script-less testing techniques attempt to alleviate the maintenance burden that comes with providing and maintaining SUT-specific information, which is a problem for intelligent monkeys. In addition, the state model and the action selection mechanism are crucial to the quality of a smart test monkey. Developing these components demands skill and a lot of work. (Kumar. 2021)

Overall, both automated GUI testing techniques have challenges that can be found summarized in the table below:

GUI Testing Technique	**Challenges**
General challenges of automated testing	Short development cycles, Increasing complexity and number of supported platforms Test oracle problem.
Script-based	High maintenance of test scripts, High manual effort to create test scripts, requires technical knowledge, requires manual effort to detect GUI widgets, requires high effort to learn the tool/programming language.
Script-less	Long execution time, Difficult reproducibility of bug’s, Smart monkey: requires technical knowledge to create a test oracle.

Table 1: Summary of the challenges of automated GUI testing
[**Source:** This thesis specific table was developed by the author.]

3.9 Research Gap

Numerous gaps and topics for more research have been found in the research on the comparison of software testing tools and methods between automated and manual testing. Research gaps in this area include the following:

Comparison between Effectiveness and Efficiency:

The efficacy and efficiency of automated versus manual testing in terms of defect discovery, test coverage, and time-to-market have been evaluated in numerous studies. More thorough and situation-specific comparisons, though, are required, taking into account variables like project size, complexity, and resource availability.

Defect Discovery:

Studies often highlight that automated testing tends to discover a higher volume of defects compared to manual testing due to its ability to execute repetitive tests with precision. However, nuanced defects that may require human intuition might be missed.

Test Coverage:

Automated testing generally provides broader test coverage as it can execute a large number of test cases quickly and consistently. Manual testing, on the other hand, might be more focused on specific areas but could miss edge cases.

Time-to-Market:

While automated testing can accelerate the testing process, manual testing might provide quicker feedback in certain scenarios, especially during early development stages or when rapid iterations are needed.

Benefit-Cost Analysis:

There is a dearth of actual data comparing the costs and advantages of manual versus automated testing across the full software development lifecycle, even though automated testing is frequently thought to be more economical in the long term. To evaluate the costs of test automation which include initial setup, maintenance, and tool licensing with the advantages of increased productivity, time savings, and quality improvement, more research is required. (Kaner, Cem 2002)

Initial Setup:

Automated testing may require significant initial setup investment in terms of tool acquisition, infrastructure, and training.

Maintenance:

While automated tests offer long-term benefits, they also require ongoing maintenance to adapt to changes in the software.

Tool Licensing:

The cost of licensing automated testing tools can vary widely and needs to be factored into the overall cost analysis.

Requirements for Training and Skills:

The importance of having competent testers and providing them with the right training for both manual and automated testing has been brought to light by research. This is done in order to ensure that there is maximum efficacy from the testers. The comparison of the training requirements, learning curves, and skill requirements for manual testers who migrate into automated testing roles, on the other hand, has not been well investigated thus far. For the aim of strengthening the skills of manual testers in automation, it is required to do more research in order to determine which training philosophies and practices are the most conducive to achieving the desired results.

Training Demands:

Transitioning from manual to automated testing requires a different skill set, including programming and familiarity with testing tools.

Learning Curves:

The time required for testers to become proficient in automated testing tools and techniques varies and can impact project timelines.

Skill Requirements:

Automation testing demands a deeper understanding of scripting languages, test frameworks, and software development principles compared to manual testing.

Test Upkeep and Development:

When it comes to automated testing, one of the most major challenges is the maintenance of tests. This is because modifications to the application that is being tested often need equivalent adjustments to test scripts and test data. One reason for this is because the program is being tested. It is necessary to conduct additional research in order to gain an understanding of the factors that influence the amount of work that is put into test maintenance. This is in addition to the development of techniques and instruments for automating test evolution, such as self-healing test scripts and adaptive test generation algorithms. (Whittaker 2008)

Script Maintenance:

Automated test scripts need to be regularly updated to reflect changes in the application's functionality or user interface.

Data Management:

In order to guarantee that the test data is accurate and relevant, it is also necessary to maintain and update.

Self-Healing Test Scripts:

It is possible that the amount of manual work that is required for the maintenance of test scripts might be greatly reduced if self-healing mechanisms are investigated. This is a possibility that is worth considering. In this regard, this would represent a significant step forward.

Usability and Human Factors:

Although automated testing has advantages over manual testing in terms of repeatability and coverage, it might not have the same human intuition and inventiveness to spot subtle flaws and usability problems. In order to create hybrid methodologies that capitalize on the advantages of both automated and human testing, research is required to examine the effects of automation on the user experience as a whole.

Repeatability vs. Intuition:

Automatic testing is particularly effective in dependably carrying out activities that are repetitive, but manual testing may make use of human intuition to uncover problems that are more complicated.

Usability Testing:

It is common for human testers to provide a more realistic assessment of the user experience, which encompasses aspects such as usability, accessibility, and the design of the user interface.

Hybrid Approaches:

It is feasible to give a better user experience and provide wide coverage by merging two separate testing approaches: automated testing and manual testing. This is achievable because of the integrated testing method.

Context-Specific Points to Remember:

Depending on the particular setting, such as the type of application (web, mobile, embedded systems), development style (e.g., agile, DevOps), and industry domain (e.g., finance, healthcare), the efficacy of human and automated testing methods and technologies may differ. To determine context-specific elements that influence the decision between automated and manual testing, as well as to produce best practices and guidelines for choosing the most appropriate approach, more research is required.

Application Type:

It is likely that some applications, such as those that have user interfaces that are exceptionally complex or that are in conformity with stringent regulatory standards, may need further testing by humans.

Development Style:

There is a possibility that automation might be advantageous to configurations that make use of Agile and DevOps in order to support continuous integration and delivery strategies.

Industry Domain:

It may be required for a corporation to use both automated and manual testing procedures in order to ensure that it is in compliance with the standards set out by different regulatory agencies. Both the healthcare industry and the banking industry are examples of sectors that have strict compliance demands.

By filling in these research gaps, we can make better decisions about software testing procedures and enhance our knowledge of the advantages and disadvantages of human and automated testing techniques and technologies. It can also aid in the creation of testing plans that are more successful and productive and that optimize the advantages of both automated and manual procedures. (Jorgensen 2013)

3.10 Summary of Literature Reviewed

Several important conclusions and insights are revealed by an examination of the literature on the comparison of manual and automated testing techniques and tools in software testing.

Efficiency and Effectiveness:

Research typically indicates that automated testing can outperform manual testing in terms of fault identification, test coverage, and time-to-market. However, the degree to which these benefits are realized may differ based on variables including project size, complexity, and resource availability.

Benefit-Cost Analysis:

Although time savings and increased productivity make automated testing seem more cost-effective in the long term, empirical data comparing the advantages and disadvantages of manual versus automated testing across the software development lifecycle is still needed. To calculate the expenses of test automation and weigh their advantages against quality enhancement and resource efficiency, more investigation is needed.

Requirements for Training and Skills:

Empirical studies emphasize the significance of proficient testers and sufficient training in manual and automated testing. Comparing the training demands, learning curves, and skill requirements for manual testers moving into automated testing roles, however, has not been thoroughly studied. The efficacy of various training modalities and approaches for up skilling manual testers in automation requires further research.

Test Upkeep and Development:

Given that modifications to the application being tested frequently necessitate matching adjustments to test scripts and test data, test maintenance becomes a major difficulty in automated testing. In addition to developing strategies and technologies for automating test evolution, such as self-healing test scripts and adaptive test generation algorithms, more study is needed to investigate the factors driving test maintenance efforts.

Usability and Human Factors:

Although automated testing has advantages over manual testing in terms of repeatability and coverage, it might not have the same human intuition and inventiveness to spot subtle flaws and usability problems. It is recommended by research to look into how automation affects the user experience overall and to create hybrid testing strategies that combine the best features of automated and manual testing.

Context-Specific Points to Remember:

Specific contexts, such as application kind, development process, and industrial domain, might influence the efficacy of human and automated testing techniques and technologies. Further investigation is required to pinpoint context-specific elements impacting the decision between automated and manual testing, as well as to create standards and best practices for determining which strategy is better.

Overall, the analysis of the literature offers insightful information about the advantages and disadvantages of automated and manual testing techniques and instruments, emphasizing the need for more study to fill in knowledge gaps and overcome obstacles in this field. It highlights how crucial it is to take human aspects and context-specific factors into account when deciding which testing strategy is best.

4. Practical Implementation

4.1 Research Design and Approach

Which method is most appropriate for implementing a set of research objectives and the circumstances under which it is conducted is determined by the design of the research. The first learning objective can be used to develop a structured framework for data collection and analysis. This study will use a well-defined research design to obtain a comprehensive understanding of the comparative advantages and disadvantages of manual and mechanical testing methods and automation.

4.1.1 Overview and Rationale

The research design chosen will provide a balanced approach, combining quantitative and qualitative data collection methods. This type of research is particularly well suited for mixed methods because it allows us to:

- **Estimating the impact of testing:** By collecting quantitative data, we are able to calculate the percentage of defects detected in each test method, the amount of time and number of subjects used for testing, and the number of tests covered.
- **Gain in-depth insights:** Qualitative data from focus groups, questionnaires and interviews can provide insightful information about test takers' perceptions and experiences of manual and automated testing this can reveal problems, perceived benefits and opportunities for improvement.
- **Improve results:** Triangulation is possible when quantitative and qualitative data are combined. By providing detailed knowledge, this enhances the validity and reliability of research findings.

There are many approaches to descriptive research, such as qualitative and quantitative methods. This research also made use of qualitative and quantitative approaches. It employs a holistic strategy. Important phases in doing research include defining the study's scope and criteria, gathering and testing participant data, and analyzing the results.

4.1.2 Mixed-Methods Approach

In order to get a thorough grasp of the phenomena being studied, this study use mixed methodologies, which integrate quantitative and qualitative research techniques. By collecting and analyzing numerical data, quantitative approaches provide statistical insights into trends, patterns, and correlations. In contrast, object-oriented approaches focus on deepening and analysing non-quantitative data to reveal the richness and complexity of human experiences.

1. **Quantitative Summary**

a. **Data Collection:** Software development teams and testers have completed structured surveys to collect quantitative data on the following topics:

- Time spent on manual testing and automation testing of tests (e.g., regression, functionality).
- Percentage of test coverage achieved in each category. Error detection rates related to automated and manual testing.
- Resource allocation (number of testers required) between projects using automation methods and projects using only manual testing.

b. **Test Management Tools:** Data on test management times, types of errors detected, and number of tests retested after issue resolution can be collected from the test management tools currently used by the teams involved in the 19th century.

c. **Data Analysis:** Quantitative data will be evaluated statistically to examine patterns, correlations, and any differences in efficiency, utility, and resource utilization between manual and automated testing.

2. Qualitative Data Collection

Data Collection Methods:

a. **Semi-Structured Interviews:** In-depth interviews with experienced hands-on experimenters will be conducted to obtain a qualitative opinion on the following.

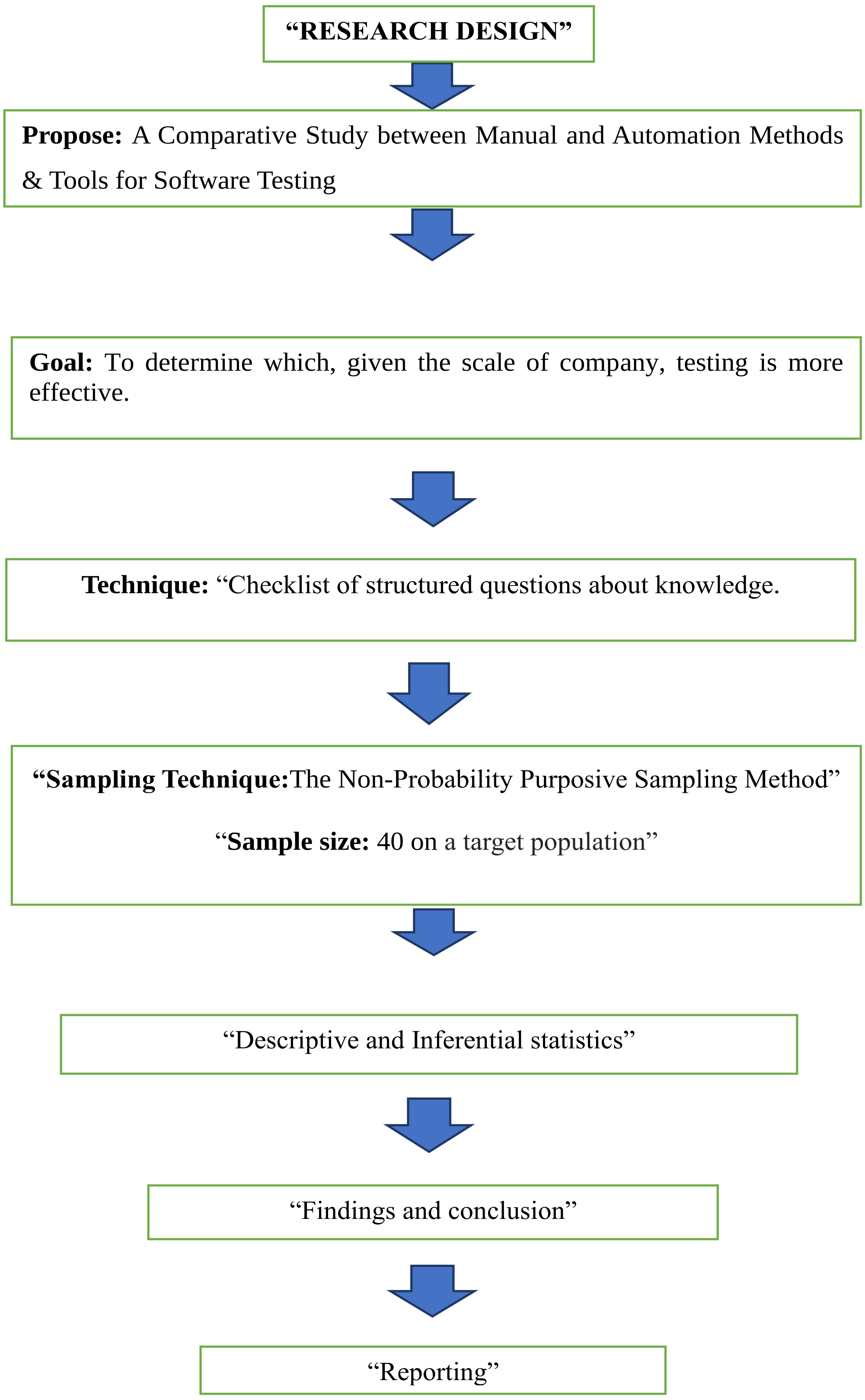
- These interviews focus on testers' perspectives on the pros and cons of each testing system.
- Difficulties with automated manual testing methods.
- Best practices and strategies to maximize the use of automated and manual testing methods.

b. **Optional Focus Groups:** Focus groups can be conducted with testers of projects using Balanced Human's automated testing approach, based on feasibility to explore:

- Communication within the team and communication between automation and manual testers.
- Strategies for effectively integrating automation and manual testing in a project.
- The effect of a combination of strategies on both the efficacy and effectiveness of a test.

c. **Data Analysis:** Analysis of the qualitative data gathered from focus groups and interviews will highlight common trends and provide crucial insights into the experiences and viewpoints of test takers as well as their exposure to all testing apparatus.

The study plan is represented diagrammatically in the diagram.



"RESEARCH DESIGN"
Propose: A Comparative Study between Manual and Automation Methods & Tools for Software Testing
Goal: To determine which, given the scale of company, testing is more effective.
Technique: "Checklist of structured questions about knowledge.
"Sampling Technique:The Non-Probability Purposive Sampling Method"
"Sample size: 40 on a target population"
"Descriptive and Inferential statistics"
"Findings and conclusion"
"Reporting"

Figure 2: Design of a Study in Schematic Form.
[**Source:** This thesis specific diagram was developed by the author.]

4.1.3 Survey Result

In-depth analysis of the interpretation of the survey data. The graph aids in a thorough examination of the data found in the table of frequencies and percentages. Analyses statistically were performed using Pearson correlation and regression.

4.1.3.1 Descriptive Statistics Frequency and Percentage of Data

In descriptive statistics, the mean, standard deviation, and standard error of the mean are displayed.

The following significant findings were discovered regarding the respondent’s demographics. Questions about basic personal information like gender, age and education come first. The success of a study depends on the researcher’s ability to accurately portray the respondent’s profile and other variables. You can see what the survey results are shown in the graphs below.

Age		
	Frequency	**Percent**
Below 25 years	9	20.0
26 to 30 years	18	40.0
31 to 35 years	7	15.6
36 to 40 years	6	13.3
Above 40 years	5	11.1
Total	45	100.0

Table 2: Age wise distribution of respondents
[**Source:** This thesis specific table was developed by the author.]

The above table discusses age wise distribution of respondents. In below 25 years group, frequency is 9 and percentage is 20%. In 25 - 30 years age group, frequency is 18 and percentage is 40%. In 31 - 35 years age group, frequency is 7 and percentage is 15.6%. In 36 – 40 years age group, frequency is 6 and percentage is 13.3%. In above 40 years age group, frequency is 5 and percentage is 11.1%.

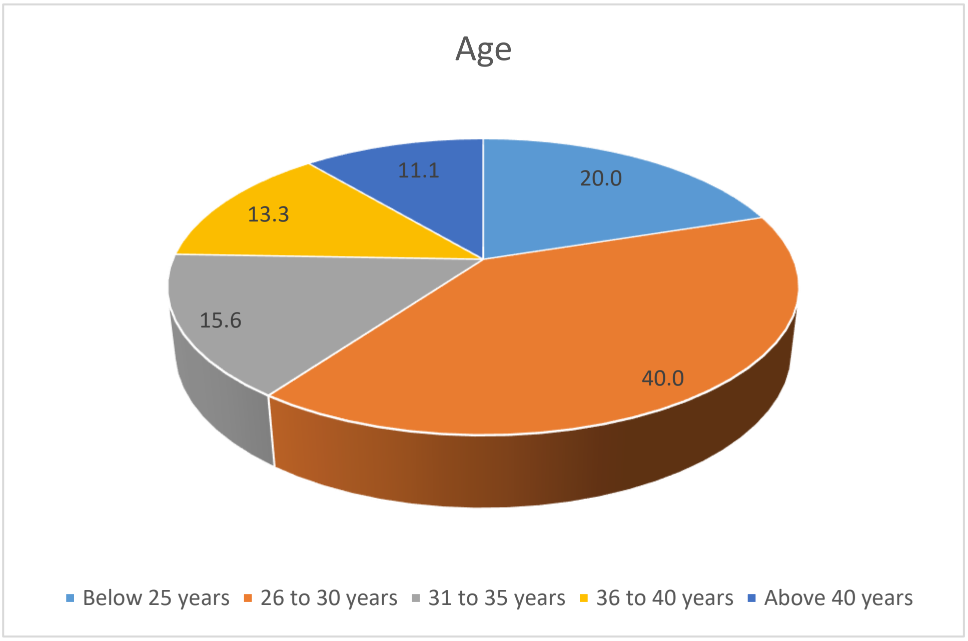


Figure 3: Graphical representation of age wise distribution of respondents.
[**Source:** This thesis specific diagram was developed by the author.]

Gender		
	Frequency	**Percent**
Male	34	75.6
Female	11	24.4
Total	45	100.0

Table 3: Gender wise distribution of respondents
[**Source:** This thesis specific table was developed by the author.]

The above table discusses gender wise distribution of respondents. In male respondents, frequency is 34 and percentage is 75.6%. In female respondents, frequency is 11 and percentage is 24.4%.

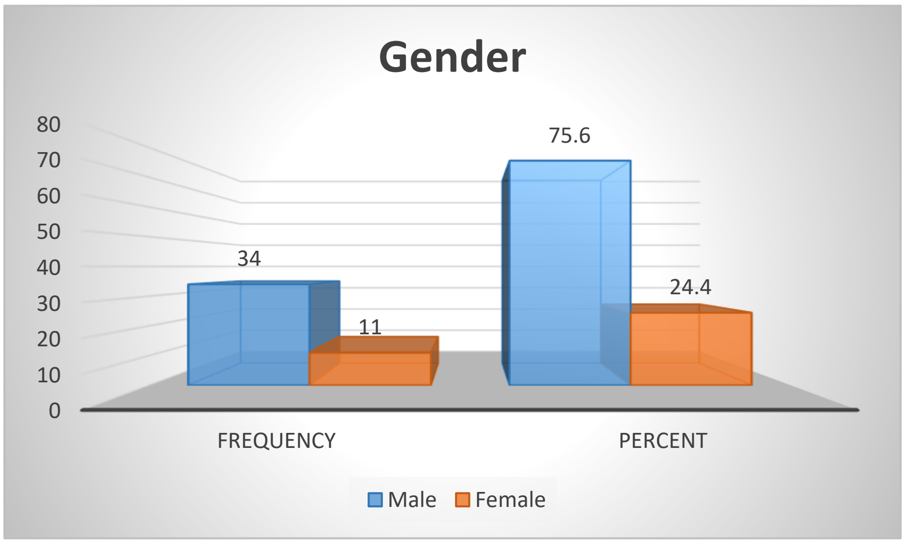


Figure 4: Graphical representation of gender wise distribution of respondents.
[**Source:** This thesis specific diagram was developed by the author.]

Years of Experience		
	Frequency	Percent
<1 year	15	33.3
1-3 years	17	37.8
3-5 years	7	15.6
5-10 years	6	13.3
Total	45	100.0

Table 4: Experience wise distribution of respondents
[**Source:** This thesis specific table was developed by the author.]

The above table discusses experience wise distribution of respondents. In below 1 year, frequency is 15 and percentage is 33.3%. In 1 – 3 years, frequency is 17 and percentage is 37.8%. In 3 – 5 years, frequency is 7 and percentage is 15.6%. In 5 – 10 years, frequency is 6 and percentage is 13.3%.

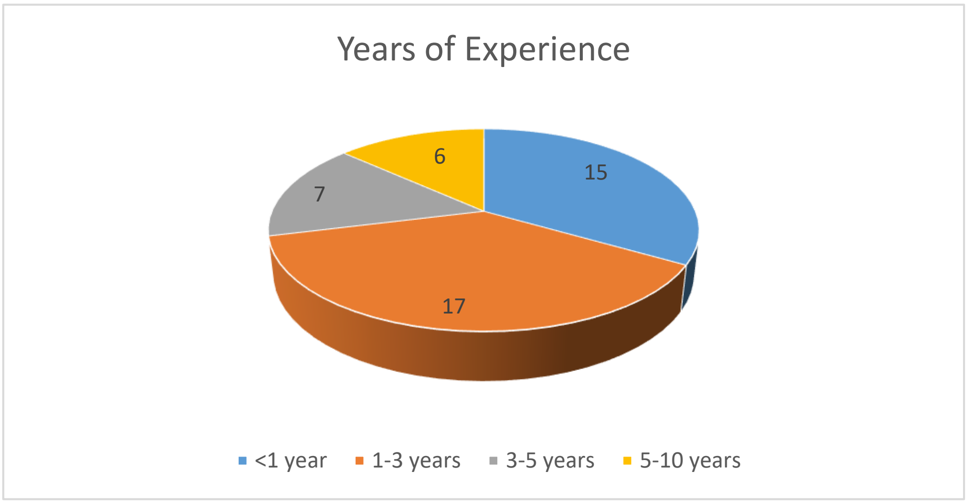


Figure 5: Graphical representation of experience wise distribution of respondents.
[**Source:** This thesis specific diagram was developed by the author.]

Job Role		
	Frequency	**Percent**
Test Analyst	9	20.0
Manual Test Engineer	12	26.7
Automation Test Engineer	6	13.3
QA Manager	7	15.6
Software Developer	6	13.3
Test Project Manager	5	11.1
Total	45	100.0

Table 5: Job role of respondents
[**Source:** This thesis specific table was developed by the author.]

The above table discusses job role of respondents. In Test Analyst, frequency is 9 and percentage is 20%. In manual test engineer, frequency is 12 and percentage is 26.7%. In Automation Test Engineer, frequency is 6 and percentage is 13.6%. In QA Manager, frequency is 7 and percentage is 15.6%. In Software Developer, frequency is 6 and percentage is 13.3%. In Test Project Manager, frequency is 5 and percentage is 11.1%.

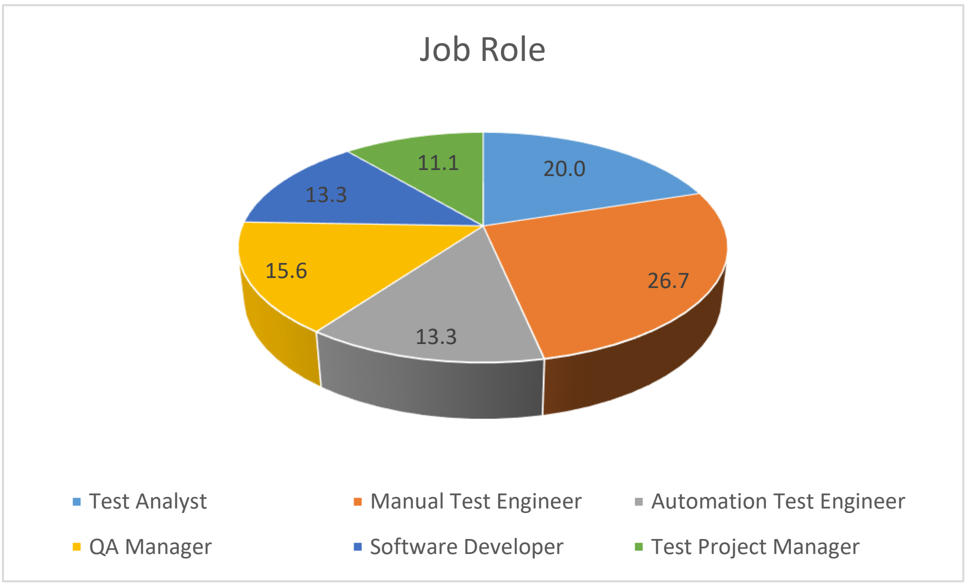


Figure 6: Graphical representation of job role of respondents.
[**Source:** This thesis specific diagram was developed by the author.]

Industry		
	Frequency	**Percent**
Finance	15	33.3
Healthcare	16	35.6
Technology	6	13.3
Manufacturing	7	15.6
Education	1	2.2
Total	45	100.0

Table 6: Industry of respondents
[**Source:** This thesis specific table was developed by the author.]

The above table discusses industry of respondents. In finance, frequency is 15 and percentage is 33.3%. In healthcare, frequency is 16 and percentage is 35.6%. In technology, frequency is 6 and percentage is 13.3% in manufacturing, frequency is 7 and percentage is 15.6%, In education, frequency is 1 and percentage is 2.2%.

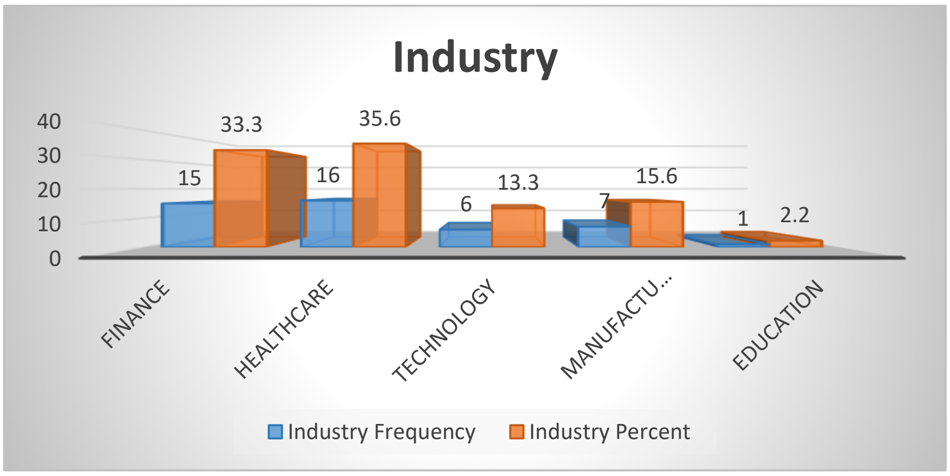


Figure 7: Graphical representation of industry of respondents.
[**Source:** This thesis specific diagram was developed by the author.]

Company Size		
	Frequency	**Percent**
Small (<50 employees)	18	40.0
Medium (50-250 employees)	22	48.9
Large (>250 employees)	5	11.1
Total	45	100.0

Table 7: Company Size
[**Source:** This thesis specific table was developed by the author.]

The above table discusses industry of respondents. In finance, frequency is 15 and percentage is 33.3%. In healthcare, frequency is 16 and percentage is 35.6%. In technology, frequency is 6 and percentage is 13.3% in manufacturing, frequency is 7 and percentage is 15.6%, In education, frequency is 1 and percentage is 2.2%.

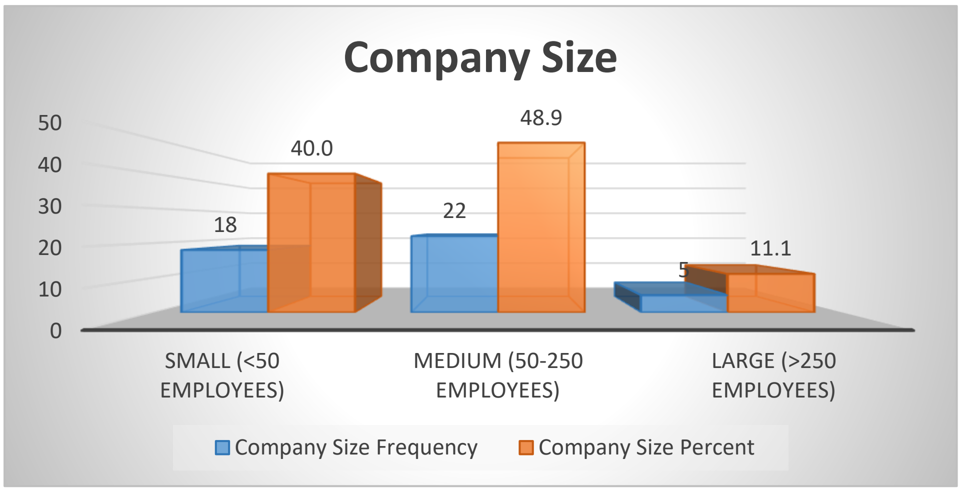


Figure 8: Graphical representation of company size.
[**Source:** This thesis specific diagram was developed by the author.]

Correlations							
		Defect Detection Rate	Test Coverage	Testing Efficiency	Cost Effectiveness	Tester Satisfaction	Quality of Testing Documentation
Defect Detection Rate	Pearson Correlation	1	.686**	.511**	.731**	.678**	.500**
	Sig. (2-tailed)		.000	.000	.000	.000	.000
	N	45	45	45	45	45	45
Test Coverage	Pearson Correlation	.686**	1	.697**	.726**	.722**	.650**
	Sig. (2-tailed)	.000		.000	.000	.000	.000
	N	45	45	45	45	45	45
Testing Efficiency	Pearson Correlation	.511**	.697**	1	.634**	.650**	.656**
	Sig. (2-tailed)	.000	.000		.000	.000	.000
	N	45	45	45	45	45	45
Cost Effectiveness	Pearson Correlation	.731**	.726**	.634**	1	.824**	.678**
	Sig. (2-tailed)	.000	.000	.000		.000	.000
	N	45	45	45	45	45	45
Tester Satisfaction	Pearson Correlation	.678**	.722**	.650**	.824**	1	.735**
	Sig. (2-tailed)	.000	.000	.000	.000		.000
	N	45	45	45	45	45	45
Quality of Testing Documentation	Pearson Correlation	.500**	.650**	.656**	.678**	.735**	1
	Sig. (2-tailed)	.000	.000	.000	.000	.000	
	N	45	45	45	45	45	45
**. Correlation is significant at the 0.01 level (2-tailed).							

Table 8: Pearson Correlation Analysis of All Variables
[**Source:** This thesis specific table was developed by the author.]

The above table discusses correlation between all the variable in which sig. value of all the variables are below 0.05 which is significant indicates all the variable are significantly correlated.

Model Summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.511[a]	.261	.244	3.45461
a. Predictors: (Constant), Testing Efficiency				

ANOVA[a]						
Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	181.136	1	181.136	15.178	.000[b]
	Residual	513.175	43	11.934		
	Total	694.311	44			
a. Dependent Variable: Defect Detection Rate						
b. Predictors: (Constant), Testing Efficiency						

Coefficients[a]						
Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	11.610	2.303		5.041	.000
	Testing Efficiency	.445	.114	.511	3.896	.000
a. Dependent Variable: Defect Detection Rate						

Table 9: Regression Analysis of Defect Detection rate and Testing Efficiency
[**Source:** This thesis specific table was developed by the author.]

Regression analysis employs ANOVA to assess the degrees of variability within a regression model and establish the foundation for a significance test. The table provided clearly indicates that the factors examined in the research exhibit statistical significance. Based on a regression analysis of testing efficiency and defect detection rate, defect detection rate has a significant impact on testing efficiency, accounting for

51% of the variation observed. The remaining portion of the variation is not accounted for and remains unexplained. The R value of 0.26 indicates a significant impact of defect detection rate on testing efficiency, as evidenced by the Anova table which shows a significant impact (F= 15.17, sign. value = 0.00). The data suggests that there is a notable correlation between defect detection rate and testing efficiency. The variable is represented in the table of coefficients above. The B-coefficients generally exhibit a positive and statistically significant relationship. Given that the dimensions of all indicators are the same, it is more advantageous to translate the B-coefficients rather than the beta coefficients. The significance value implies that there is significant effect of defect detection rate on testing efficiency.

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.697[a]	.485	.473	2.47279

a. Predictors: (Constant), Testing Efficiency

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	248.047	1	248.047	40.566	.000[b]
	Residual	262.931	43	6.115		
	Total	510.978	44			

a. Dependent Variable: Test Coverage

b. Predictors: (Constant), Testing Efficiency

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	10.344	1.649		6.274	.000
	Testing Efficiency	.520	.082	.697	6.369	.000

a. Dependent Variable: Test Coverage

Table 10: Regression Analysis of Test Coverage and Testing Efficiency
[**Source:** This thesis specific table was developed by the author.]

Regression analysis employs ANOVA to assess the degrees of variability within a regression model and establish the foundation for a significance test. The table provided clearly indicates that the factors examined in the research exhibit statistical significance. Based on a regression analysis of testing efficiency and test Coverage, Test Coverage has a significant impact on testing efficiency, accounting for 69% of the variation observed. The remaining portion of the variation is not accounted for and remains unexplained. The R value of 0.48 indicates a significant impact of Test Coverage on testing efficiency, as evidenced by the Anova table which shows a significant impact (F= 40.56, sign. value = 0.00). The data suggests that there is a notable correlation between Test Coverage and testing efficiency. The variable is represented in the table of coefficients above. The B-coefficients generally exhibit a positive and statistically significant relationship. Given that the dimensions of all indicators are the same, it is more advantageous to translate the B-coefficients rather than the beta coefficients. The significance value implies that there is significant effect of test coverage on testing efficiency**.**

4.1.4 Interview Schedule

Expert 1 (5 years, Web application testing):

- I have 5 years of experience in web application testing, with a particular emphasis on assessing functionality and usability.
- Approximately 70% of projects utilize a combined approach. Manual testing is essential for conducting initial exploration and evaluating user experience, but automation simplifies the process of regression testing.

Expert 2 (10 years, Mobile app testing):

- I have accumulated over 10 years of expertise, mostly focusing on testing mobile applications for both iOS and Android platforms.
- Approximately 80-90% of mobile app initiatives derive advantages from employing a combination of approaches strategy. Automation is highly effective in doing repetitive activities, but manual testing continues to be essential for evaluating usability and handling edge-case problems.

Expert 3 (8 years, API testing):

- I have accumulated 8 years of professional experience, with a specific focus on API testing and backend functionality.
- The majority of API testing projects (over 95%) significantly rely on automation because API requests are often repetitive in nature. Manual testing is centered around the integration of different components and the examination of specific scenarios or use cases.

Expert 4 (12 years, Performance testing):

- I have accumulated 12 years of expertise in the field of performance testing and load optimization.
- The composition may differ; however, the majority of performance testing typically consists of a mix, including around 60-70%. Manual testing is useful for identifying bottlenecks, but automation is helpful for doing load simulation and scalability analysis.

Expert 5 (7 years, Security testing):

- Seven years of experience conducting security testing, with a focus on penetration testing and vulnerability identification.
- A common method is essential (80–90 percent). Exploitation strategies that are not conventional require manual testing, whereas regression testing and vulnerability scanning are facilitated by automation.

Expert 6 (Advanced Manual Techniques):

- Nine years of experience, with a solid background in both automated and manual testing.
- The split varies, but it’s important to comprehend both strategies well. For comprehensive test case design, I regularly employ boundary value analysis and equivalency partitioning.

Expert 7 (Automation Tool Selection):

- I have accumulated 8 years of professional experience, specializing in automation frameworks and technologies.
- The selection of tools is determined by the project. Open-source alternatives like as Selenium are widely used, however for intricate projects, it may be essential to employ commercial products that offer advanced functionality. Compatibility with scripting languages is a significant factor to take into account.

Expert 8 (Efficiency gains with Automation):

- I have accumulated over 10 years of expertise and possess a strong enthusiasm for utilizing automation to enhance the efficiency of testing processes.
- Automation has greatly enhanced the efficiency of regression testing in several projects; however the exact proportion may vary. I have employed many technologies to mechanize monotonous operations, therefore liberating time for exploratory testing.

Expert 9 (Challenges in Automation):

- 12 years of experience and a thorough awareness of the advantages and difficulties associated with automation.
- The combination is specific to a project. Two major issues in automation are handling flaky tests and maintaining test scripts.

Expert 10 (Balancing Automation & Agile):

- 7 years of experience and proficiency with agile testing techniques.
- It is essential to maintain balance. In agile sprints, we emphasize manual testing for new features while striving for good automation coverage.

Future of Testing (All Experts Agree):

- Artificial intelligence (AI) and machine learning (ML) have the capacity to transform testing processes by automating operations such as generating test cases, detecting anomalies, and even creating self-repairing test scripts.

- Testers must acquire and cultivate expertise in domains like as AI, data science, and analytical thinking in order to effectively collaborate with these emerging technologies.

Conclusion

The Interview of experts' views on manual and automated testing methodologies uncovers many significant findings, A combination of manual and automation testing is the most efficient technique for the majority of software testing projects, but the specific percentages may differ depending on the testing area. Proficiencies encompass doing usability testing, performing exploratory testing, and effectively managing edge situations. One weakness is that it might be time-consuming for repetitive activities. Significant strengths including the ability to efficiently handle repeated jobs, do regression testing, and perform API testing. Weaknesses encompass the burden of maintenance and the difficulties associated with unreliable tests. The most effective strategy varies depending on the individual requirements of the project. Gaining a comprehensive understanding of the advantages and disadvantages of each approach is essential for making well-informed choices. Equivalence partitioning and boundary value analysis are effective techniques for meticulous test case creation in manual testing. When selecting automation technologies, factors such as project complexity, scripting languages, and money are all influential. Open-source alternatives are widely favored, while complex tasks may necessitate the use of commercial software. The automation of operations such as test case creation and self-healing scripts is anticipated to transform testing through the use of AI and machine learning. To effectively collaborate with these emerging technologies, testers must acquire and cultivate proficiencies in AI, data science, and analytical reasoning. In conclusion, attaining the highest level of test coverage necessitates a deliberate amalgamation of human and automation testing techniques. Testers may enhance the thoroughness and efficiency of the testing process by utilizing the advantages of each technique and acknowledging their limits. With the changing testing landscape, the advancements in AI and machine learning have the potential to improve testing capabilities. Testers must acquire new skillsets to remain at the forefront of this progress.

4.2 Comparative Analysis Framework

4.2.1 Criteria for Comparison

The criteria for comparison inside the Comparative Analysis Framework are the precise dimensions or elements underneath which the developed software may be evaluated. These standards should be carefully decided to make a fair, comprehensive, and relevant evaluation. The choice of standards depends at the context of the assessment and the goals of the analysis.

- **Effectiveness:** Measures how well a software program achieves its meant results.
- **Efficiency:** Assesses assets needed to acquire favored results.
- **Scalability:** Assesses the software program's potential to address increasing work scopes without compromising overall performance.
- **Usability:** Evaluates the software program's person-friendliness, accessibility, and intuitiveness.
- **Flexibility and Adaptability:** Assesses the software program's capability to alter to modifications or demanding situations.
- **Sustainability:** Assesses the software's lengthy-term viability and environmental impact.
- **Return on Investment (ROI):** Evaluates the profitability or value-effectiveness of the software.
- **Quality:** Assesses the overall excellence of a software entity.
- **Security:** Evaluates the software program's robustness against threats and vulnerabilities.
- **Innovation and Creativity:** The software's capacity to introduce novel answers or improvements.

The above-stated criteria cover all of the relevant components of the software program being in comparison, measurable or assessable in a scientific way, and align with the desires and priorities of the business stakeholders involved within the analysis.

4.2.2 Metrics for Effectiveness and Efficiency

To quantitatively assess the effectiveness and performance of manual versus automated software program testing methods, we can compare them throughout several key metrics. These metrics consist of setup time, execution time, price, accuracy, scalability, flexibility, repeatability, and the talent level required. This evaluation will be segmented by using brief-scale, medium-scale, and large-scale IT industry projects over brief-term, medium-term, and long-time period periods. The assessment is primarily based on standard developments found in the industry. The unique values may additionally vary relying on the exact nature of the project and the tools or methodologies used. The aim right here is to offer a broader assessment.

Project Scale:

- **Short-Scale Projects:** Small teams and scope, commonly lasting some weeks.
- **Medium-Scale Projects:** Medium-sized groups and scope, generally lasting a few months.
- **Large-Scale Projects:** Large groups and good sized scope, regularly lasting six months or greater.

Project Duration:

- Short-Term: Up to 3 months.
- Medium-Term: 3 to 6 months.
- Long-Term: More than 6 months.

Metric	**Manual Testing (Short-Term)**	**Automated Testing (Short-Term)**	**Manual Testing (Medium-Term)**	**Automated Testing (Medium-Term)**	**Manual Testing (Long-Term)**	**Automated Testing (Long-Term)**
Setup Time	Low to Medium	High	Low to Medium	High	Low to Medium	High
Execution Time	High	Low	High	Low	High	Low
Cost	Low	High (initial)	Medium	Medium (over time)	High	Lower (over time)
Accuracy	Medium	High	Medium	High	Medium	High
Scalability	Low	High	Low	High	Low	High
Flexibility	High	Medium	High	Medium	High	Medium
Repeatability	Low	High	Low	High	Low	High
Skill Level Required	Medium	High	Medium	High	Medium	High

Table 11: Quantitative Analysis to achieve Test Effectiveness & Efficiency. [**Source:** This thesis specific table was developed by the author.]

Metrics:

- **Setup Time:**

  Is the time required to put together and begin the trying out process.

  Automated checking out requires greater preliminary setup, especially for short-time period projects. However, this investment will pay off in longer-term initiatives.

- **Execution Time:**

  How lengthy it takes to finish the trying out cycle.

  Automated checking out appreciably reduces execution time, which becomes more useful as project size and length increase.

- **Cost:**

  Initial and ongoing expenses associated with the testing process.

  Manual trying out may additionally appear less costly inside the quick time period because of lower preliminary setup expenses. However, automated checking out will become extra cost-powerful through the years because of reduced execution instances and the want for fewer human resources.

- **Accuracy and Scalability:**

  The reliability of the checking out method in identifying defects is determined as Accuracy and the capability to deal with increasing quantities of labor or being able to be enlarged is described as Scalability.

  Automated testing is more correct and scalable, making it specially perfect to huge-scale and lengthy-term initiatives.

- **Flexibility and Repeatability:**

  Ease of adapting the trying out manner to adjustments in venture scope or technologies is referred as flexibility. Consistency of trying out results over a couple of cycles is repeatability.

  Manual trying out offers greater flexibility however lacks the repeatability of automatic assessments, that can continually execute the same exams with high precision.

- **Skill Level Required:**

  Is the understanding needed to carry out testing successfully.

  Automated testing calls for a better talent level for setup and renovation of check scripts, while manual testing requires information of the software under check but less technical ability in scripting.

This contrast highlights that at the same time as computerized testing involves better initial setup time and fees, its blessings in execution time, accuracy, scalability, and repeatability make it a more effective and efficient approach for medium to large-scale tasks over the medium to long time. Manual checking out stays treasured for its flexibility and decrease initial value, particularly in short-term, small-scale projects in which the overhead of automation may not be justified. As tasks grow in scope and length, the investment in automated testing can result in substantial enhancements in efficiency and effectiveness, in spite of the better talent stage required to put in force and maintain the testing framework.

4.3 Testing Processes and Techniques

4.3.1 STLC and SDLC Integration

Software Development Life Cycle Method (SDLC)

SDLC abbreviated as Software Development Life Cycle is a systematic process that includes various phases of software development and the order of execution of phases namely Planning, Defining, Designing, Development, Testing and Deployment and Maintenance phase. SDLC creates the structure of development of software and each phase requires deliverables from the previous phase in SDLC.

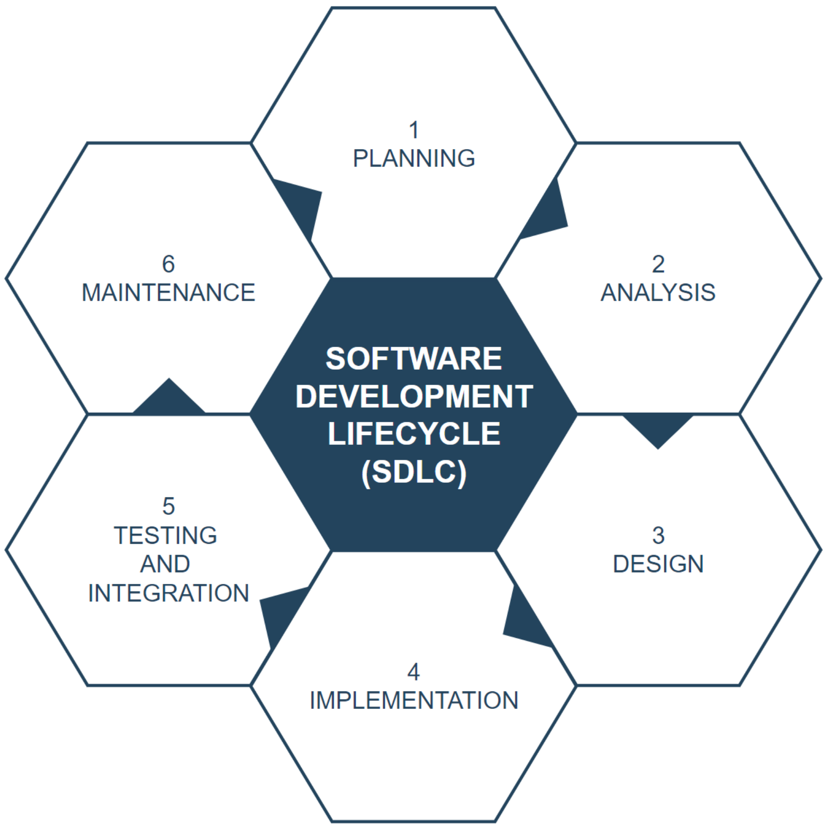


Figure 9: Software Development Life Cycle.
[**Source:** This thesis specific diagram was developed by the author.]

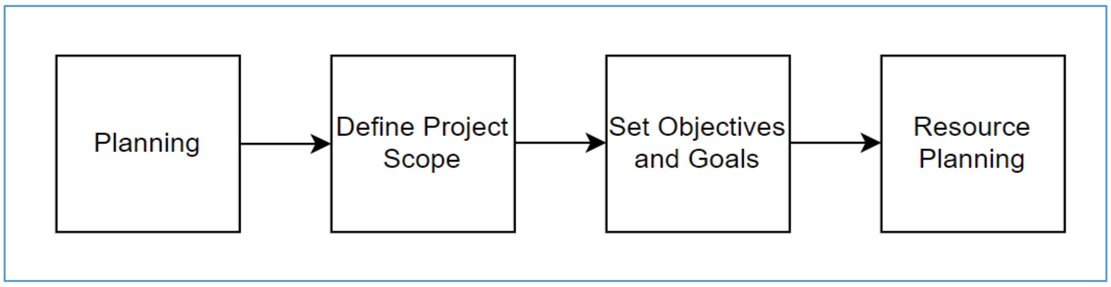


Figure 10: Planning and Requirement Analysis Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 1: Planning and Requirement Analysis**

Planning is the crucial step in software development. In this initial phase, project stakeholders define the scope of the project, set goals, and determine the resources and timelines. The quality of the software/application is a result of planning phase. Hence key activities include feasibility studies, risk assessment, and project planning.

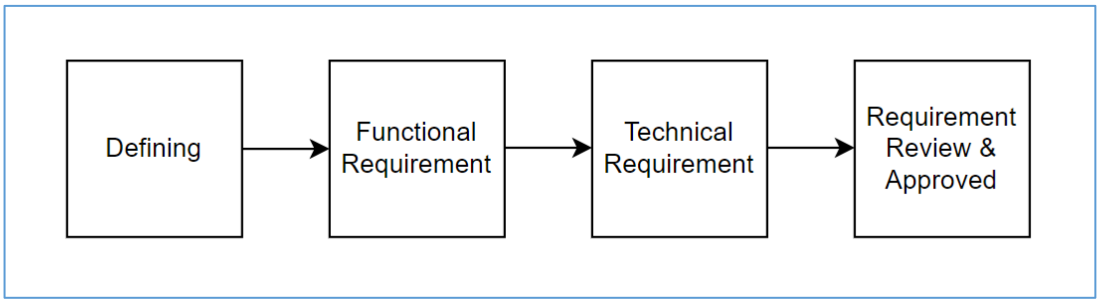


Figure 11: Defining Requirements Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 2: Defining Requirements**

During requirement gathering and elicitation phase, the project team works closely with stakeholders to gather and analyze requirements. The goal is to understand the needs of users and define the functional and non-functional requirements of the software.

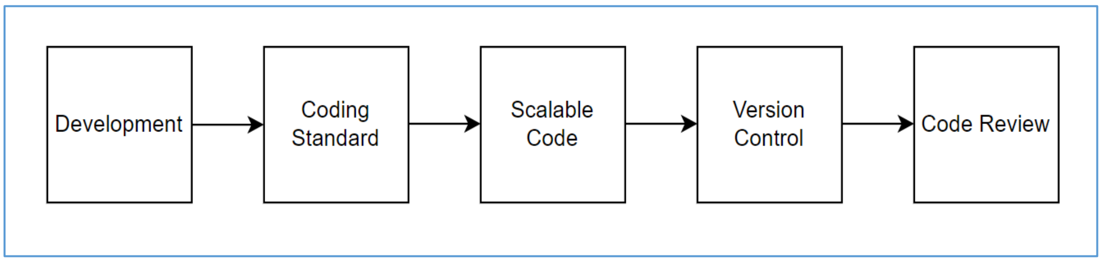


Figure 12: Designing Architecture Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 3: Designing Architecture**

Software requirement specification (SRS)/ Customer requirement specification (CRS) is a reference document for software designers to produce best architecture. Architects and designers develop system architecture, data structures, user interfaces, and other design elements. The design phase involves creating a detailed blueprint of the software based on the gathered requirements in Design document specification (DDS). The DDS is evaluted by market analysts and stakeholders. After evaluating all the factors, the most practical and logical design is chosen for development.

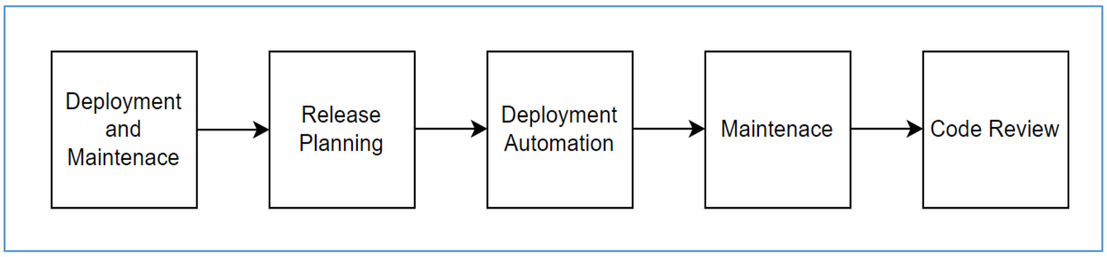


Figure 13: Developing Product Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 4: Developing Product**

During the implementation phase, developers write the software's actual code based on the design specifications. Coding standards are followed, and the code is typically reviewed for quality and adherence to best practices.

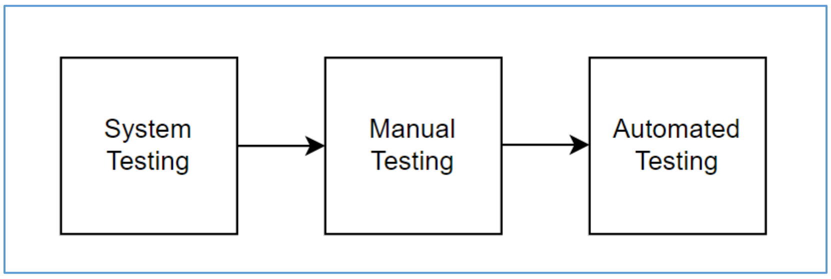


Figure 14: Product Testing and Integration Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 5: Product Testing and Integration**

Once the product is developed, testing of the software is required to ensure that all the software functions as intended and meet the specified requirements. Testers conduct various types of testing including unit testing, integration testing, system testing, regression testing and user acceptance testing.

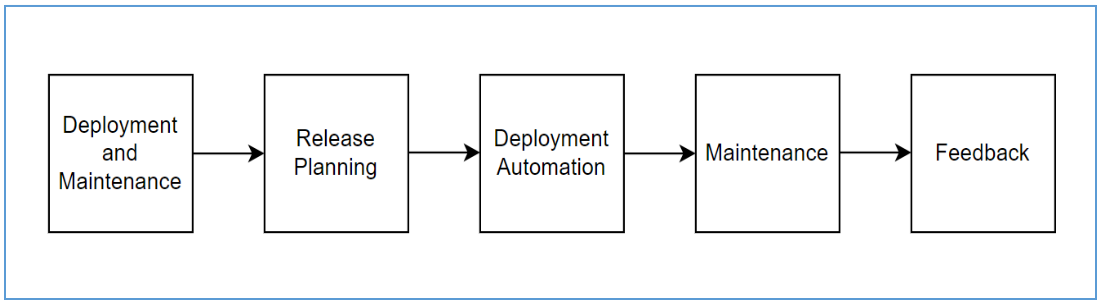


Figure 15: Deployment and Maintenance of Products Stage.
[**Source:** This thesis specific diagram was developed by the author.]

- **Stage 6: Deployment and Maintenance**

Once the software has been properly tested and approved, it is moved to the production environment. Data migration, installation, and configuration are some examples of deployment activities. Following deployment, the program enters the maintenance phase, during which continuing support and upgrades are provided. Bug fixes, product enhancements, and other changes are performed in response to business needs.

These phases can be executed in a linear or iterative fashion, depending on the chosen SDLC model. The SDLC process provides a structured approach to software development, guiding teams through different stages, ultimately aiming to deliver high-quality software that meets user expectations.

Software Testing Life Cycle Technique (STLC)

Procedure that is used in Software testing is known as Software Testing Life cycle is a structured approach that includes a series of well-defined phases of testing process from requirement analysis to test closure that ensures quality, reliability and functionality of a software applications. By following the well-defined phases of STLC that are mentioned below, software development teams can identify defects early, improve software quality, and deliver robust solutions that meet user expectations.

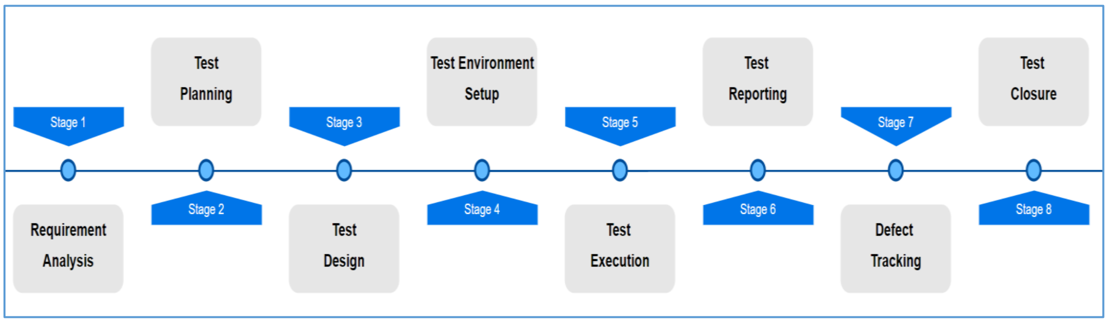


Figure 16: Software Testing Life Cycle.
[**Source:** This thesis specific diagram was developed by the author.]

- **Phase 1:** In the first phase of a project, requirements and specifications are thoroughly analyzed. The testing team works closely with business analysts to understand the software's intended functionality. By defining the testing scope and identifying test scenarios, this phase prepares the ground for effective testing.

- **Phase 2:** Test planning is an essential step in ensuring a thorough and effective testing process. It involves the development of a detailed test plan that outlines the strategy, objectives, resources, schedule, and expected results of the testing process. This plan serves as a roadmap for the entire testing process, providing a clear understanding of the scope of testing and the environment required to execute it. By carefully planning and executing the testing process, organizations can optimize their testing efforts, allocate resources efficiently, and manage their timelines effectively, ultimately leading to higher quality products and improved customer satisfaction.

- **Phase 3:** During the test case design phase, detailed test cases are developed based on the requirements and test objectives outlined in the test plan. These test cases outline specific steps to be taken, input data and expected results,. The meticulous creation of test cases enables thorough coverage of the software's functionalities, resulting in accurate validation and bug detection.

- **Phase 4:** Establishing a testing environment that closely resembles the production environment is crucial for effective testing. This process involves setting up the necessary hardware, software, databases, network connections, and other elements required to simulate real-world scenarios. A well-designed test environment can help ensure that testing accurately reflects how the system will perform in real-world use.

- **Phase 5:** During the test execution phase, the developed test cases are executed on the software in a systematic manner. The testing team records the outcomes of these tests and compares them to the expected results. The primary objective of this phase is to identify any defects, inconsistencies, or deviations from the desired behavior. By conducting thorough testing during this phase, issues are identified early on, which helps to reduce the cost of fixing them later on.

- **Phase 6:** In defect reporting and tracking phase, when defects are discovered during test execution, they are reported in a defect tracking system. Each defect is assigned a unique identifier, its severity and priority are determined, and it is tracked until it is resolved. Defect reporting and tracking guarantee that identified issues are addressed and documented for future reference.

- **Phase 7:** Test reporting involves generating various reports to communicate the progress, test results, and defect status to stakeholders. These reports provide insights into the software’s quality, highlight potential risks, and help project managers make informed decisions about the software’s readiness for release.

- **Phase 8:** Upon completion of the test execution phase, a formal closure process ensures that all necessary criteria have been fulfilled. The testing team assesses the extent to which the exit criteria for testing have been met and produces a comprehensive test summary report. This document provides an overview of the testing activities, their outcomes, and any valuable lessons learned throughout the testing process.

4.3.2 Comparative Analysis of Testing Processes

Compiling a comparative analysis of testing procedures involves organizing data by different aspects like testing types, purpose, scope, techniques, tools, and involved stakeholders. An example is presented comparing Manual Testing and Automated Testing, which can be expanded to cover additional testing types or dimensions for a thorough analysis. This simplification may overlook details related to various development methodologies or organizational practices. Adapting the comparison to suit the project or organizational needs is crucial.

Criterion	**Manual Testing**	**Automated Testing**
Purpose	To manually execute test cases without using any automation tools.	To use automation tools to execute test cases without human intervention.
Scope	Ideal for exploratory, usability, and ad-hoc testing where human observation is crucial.	Best suited for regression, load, and performance testing, where repetitive and extensive tasks are involved.
Techniques	Black-box testing, White-box testing, Grey-box testing.	Scripted testing, Data-driven testing, Keyword-driven testing.
Tools	Test management tools (e.g., JIRA, TestRail), Documentation tools (e.g., Confluence).	Automation frameworks (e.g., Selenium, QTP), CI/CD tools (e.g., Jenkins, GitLab CI).
Execution Time	Time-consuming due to manual effort required for each test case.	Significantly faster after initial setup, as tests can be run automatically at any time.
Cost	Lower initial cost but higher long-term cost due to ongoing manual effort.	Higher initial cost for setup and maintenance, but lower long-term cost due to reusability and scalability.
Accuracy	Subject to human error, but beneficial for detecting visual and usability issues.	High accuracy for detected failures in test cases, but may miss visual and usability issues.
Flexibility	High flexibility to adapt to changes in the application or testing requirements.	Requires updates to test scripts when application changes, which can be time-consuming.
Stakeholders Involved	Testers, QA analysts, sometimes end-users for UAT (User Acceptance Testing).	Testers, Developers (for writing and maintaining scripts), DevOps (for integrating with CI/CD pipelines).
Best Used For	Early stages of development, small to medium projects, features requiring human judgment.	Large scale projects, projects with long maintenance phases, areas requiring frequent regression testing.

Table 12: Comparative Analysis of Manual vs. Automated Software Testing
[**Source:** This thesis specific table was developed by the author.]

A detailed examination of different testing techniques such as Smoke, Sanity, Functional, Regression, Retesting, Unit, Integration, System, and User Acceptance Testing (UAT), and how they align with manual and automation testing methods is needed to create a comparative analysis. A comprehensive comparison in table format is available.

Testing Technique	**Purpose**	**Scope**	**Manual Testing Compatibility**	**Automation Testing Compatibility**
Smoke Testing	First testing performed on newly released initial build (unstable) in order to check the basic & critical functionalities. And the deployed software build is stable or not?	Narrow scope, focusing on critical functionalities of the software.	High, especially for initial builds.	High, suitable for automated smoke tests for frequent builds.
Sanity Testing	Is performed on stable build in order to perform deep testing on selected functionalities. To check minor changes or fixes have not affected existing functionalities in a new build.	Narrow, focusing on specific components or functionalities affected by recent changes.	High, for quick checks without detailed scripts.	Moderate, automation can be used for repetitive sanity tests.
Functional Testing	Is only concerned with validating if a system works as intended. And the ultimate goal of functional testing is to ensure that software works according to specifications and user expectations.	Broad, covering all functionalities of the application.	High, due to the need for varied human interaction.	High, especially for regression functional testing.

Regression Testing	To ensure that new changes or modifications have not adversely affected existing functionalities.	Broad, as it involves re-testing the entire application.	Moderate, for selective critical areas.	Very High, ideal for automation due to repetitive nature.
Retesting	Is a type of software testing where specific test cases that previously failed or identified defects are executed again after the defects have been fixed or the code has been modified and also to ensure that the fix hasn't introduced new defects in unchanged areas of the software.	Specific to defects that were identified and supposed to have been fixed.	High, to verify specific bug fixes.	Moderate, automated scripts can be used for known bug fixes.
Unit Testing	To test individual units or components of the software. The purpose of this testing is to check whether each module is working properly	Very narrow, focusing on the smallest testable parts of an application, like functions or methods.	Moderate, mainly for complex logic that's hard to automate.	Very High, most suitable for automation.
Integration Testing	To test the integration or interfaces between components, or between different systems. To check modules are communicating each other as Data Flow Diagram specified in Technical Document.	Moderate, focusing on the interactions between integrated components or systems.	Moderate, to check the flow of data and control.	High, automation can facilitate testing of numerous integration points.

System Testing	To validate the complete and integrated software product.	Broad, covering the software entirely to evaluate its compliance with the requirements.	High, for overall system evaluation.	Moderate to High, automation can be used but may not cover all aspects.
User Acceptance Testing (UAT)	To ensure the software can handle real-time user requirements and is ready for deployment in client's environment	Broad, simulating real-world usage and scenarios.	Very High, requires end-user experience and feedback.	Low, manual testing is preferred to assess user satisfaction.

Table 13: Comparison of Software Testing Techniques Compatibility.
[**Source:** This thesis specific table was developed by the author.]

Manual and Automation Testing Methods are well matched in distinctive ways. Manual Testing is only for techniques that need human inputs like UAT, Sanity, and System Testing. On the alternative hand, Automation Testing excels in repetitive, data-driven, or regression based processes which include Regression, Unit, and Integration Testing.

4.4 Tools and Technologies Implementation

4.4.1 Automation Testing Tools Overview

Automation testing tools are critical in software development, allowing groups to test the functionality, reliability, performance, and security in their applications efficiently and accurately. These tools automate the running of test cases, eliminating manual testing, which can be time-consuming and error prone. The selection of an automation testing tool depends on factors such as the type of software application, programming languages, testing sophistication level, and budget.

Here is an overview of various categories and tools in the automation testing field. Functional Testing Tools are used to test software functionality and ensure it behaves as expected by simulating user actions and verifying outputs. Selenium is an open-source tool for web testing supporting multiple browsers and languages. Appium is for mobile apps like iOS and Android. UFT, previously QTP, is from Micro Focus and supports desktop, web, and mobile apps. Performance Testing Tools evaluate speed and stability, including JMeter for performance and LoadRunner for simulating multiple users. CI/CD Tools like Jenkins automate building, testing, and deploying apps. Security Testing Tools like OWASP ZAP and Fortify help identify vulnerabilities and protect against cyber threats. API testing tools are vital for modern applications that heavily rely on APIs for functionality, focusing on testing APIs directly. Postman is a popular tool for sending requests and receiving responses from a web server, while SoapUI is specifically for testing SOAP and REST APIs. Mobile testing tools ensure seamless performance across devices and operating systems, with Espresso for Android and XCTest for iOS and macOS.

The automation testing tool landscape is diverse and constantly evolving, with options for functionality, performance, security, and API testing. Effective tool selection is key for improving software quality, reducing testing time, and increasing development efficiency.

4.4.2 Criteria for Tool Selection

Criteria	Description	Importance
Compatibility	The tool needs to be compatible with the platforms, operating systems, and technologies utilized in the application.	• High • Ensures that the tool can test the application effectively across the required platforms.
Ease of Use	The learning curve and ease of setting up and using the tool. Includes the availability of a user-friendly interface and documentation.	• Medium to High • Affects the speed of adoption and productivity of the testing team.
Integration Capabilities	The ability to integrate with other tools and systems in the development pipeline, such as CI/CD tools, version control systems, and project management tools.	• High • Critical for enabling a seamless and automated workflow throughout the development and testing phases.
Support for Automation Feature	The variety and complexity of automation functionalities offered such as codeless automation, script reusability, and data-driven testing.	• High • Determines the efficiency and flexibility of creating and managing tests.
Performance and Scalability	The tool's ability to handle many tests at speed and scale up to accommodate growing project needs.	• High • Essential for ensuring the tool remains viable as the project and its testing requirements grow.
Community and Support	The presence of both a strong community support system and professional assistance from the tool vendor can play a vital role in problem-solving and skill development.	• Medium • Can help resolve issues more quickly and enhance the tool's usability.

Cost	The total cost of ownership, including licensing fees, training costs, and any additional expenses for updates/support should align with project's support	• Medium to High • A key consideration for most projects, especially those with limited budgets.
Test Defect Reporting and Analytics	The reporting and analytics capabilities are top-notch, providing in-depth insights into test coverage, defects, and performance trends through detailed test reports.	• Medium to High • Crucial for understanding test outcomes, identifying trends, and making informed decisions.
Security Features	The tool's features and protocols for ensuring the security of test data and the testing environment.	• Medium to High • Particularly important for applications dealing with sensitive data or in regulated industries.

Table 14: Software Testing Tool Selection Criteria
[**Source:** This thesis specific table was developed by the author.]

4.4.3 Comparative Evaluation of Tools

A comparative evaluation of various automation testing tools across key aspects will help in understanding their strengths, weaknesses, and ideal use cases. The tools selected for comparison cover a range of testing needs, including web and mobile application testing, performance testing, CI/CD, security testing, API testing, and UI testing for mobile apps.

Feature/ Criteria	**Selenium**	**Appium**	**JMeter**	**Jenkins**	**Postman**
Type of Testing	Functional (Web)	Functional (Mobile)	Performanc e	CI/CD	API
Open Source	Yes	Yes	Yes	Yes	No (Free and Paid versions)
Platform Support	Web browsers	Android, iOS	Web applications	Multiple platforms	API platforms
Programming Languages	Java, C#, Ruby, Python, others	Java, Ruby, Python, others	Java	Groovy, any via plugins	-
Ease of Use	Moderate-High (depends on setup)	Moderate (requires setup)	Moderate	Moderate-High (depends on setup)	Easy
Integration	High (with other testing tools)	High (with other testing tools)	Moderate-High	Very High	High
Community Support	Very High	High	High	Very High	Very High
Reporting	Moderate	Moderate	High	High	High
Cost	Free	Free	Free	Free	Free (with paid options)

Table 15: Comparison of Software Testing Tools Across Various Criteria
[**Source:** This thesis specific table was developed by the author.]

4.5 Test Case Design and Execution

4.5.1 Importance of Test Cases

Test cases are an imperative a part of software checking out, forming the basis for powerful trying out strategies and making sure software excellent, reliability, and performance thru structured trying out of anticipated behaviours, potential errors, and part instances. The significance of check cases is summarized with exceptional components:

- **Specification of Testing Objectives:** Test instances without a doubt state the desires of checking out for every function or aspect, outlining what will be tested and the anticipated results. This aids in understanding the trying out scope and achievement criteria.
- **Ensuring Comprehensive Coverage:** Well-crafted check cases cowl all software program functionalities, which include high quality, negative, and edge instances. This is important for figuring out person enjoy issues and device screw ups.
- **Facilitating Automated Testing:** Test instances are crucial for automatic testing, guiding test execution scripts for constant and green trying out methods.
- **Reproducibility of Defects:** Test cases allow defects to be replicated through step-via-step commands, assisting builders in fixing problems and testers in verifying the effectiveness of the fixes.
- **Regression Testing:** Test cases are key for retesting software after changes, making sure that updates do no longer effect existing functionalities.
- **Benchmarking and Quality Assurance:** Test cases act as satisfactory benchmarks, enabling teams to tune progress and make sure software program meets satisfactory standards earlier than launch.
- **Documentation and Knowledge Transfer:** Test cases document checking out approaches and effects, serving as a knowledge base for current and destiny checking out efforts, facilitating new group member onboarding and retaining checking out consistency.
- **Legal and Compliance Assurance:** In some sectors, take a look at cases are wanted to reveal adherence to criminal and regulatory requirements, serving as

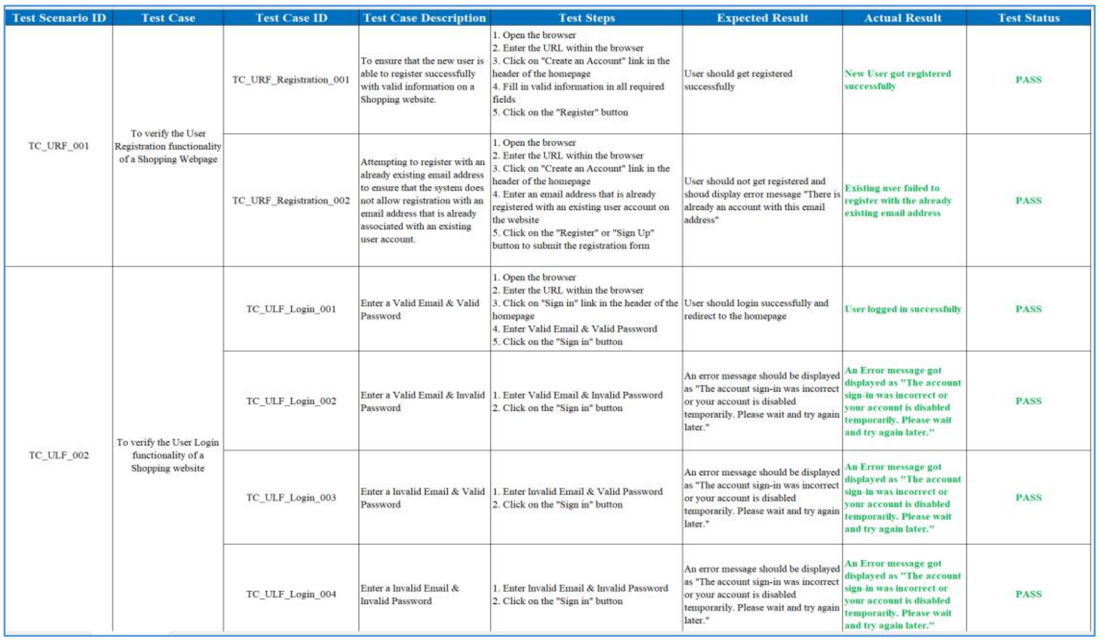
proof that the software has been adequately examined and meets specific necessities.

Therefore, check instances play a vital function within the trying out procedure by using imparting guidance, ensuring complete insurance, enabling automation, supporting illness reproducibility, facilitating regression testing, documenting trying out sports, and ensuring adherence to excellent requirements. Their systematic method in defining standards and anticipated effects is essential for delivering top-notch software program merchandise.

4.5.2 Manual and Automation Test Case Design

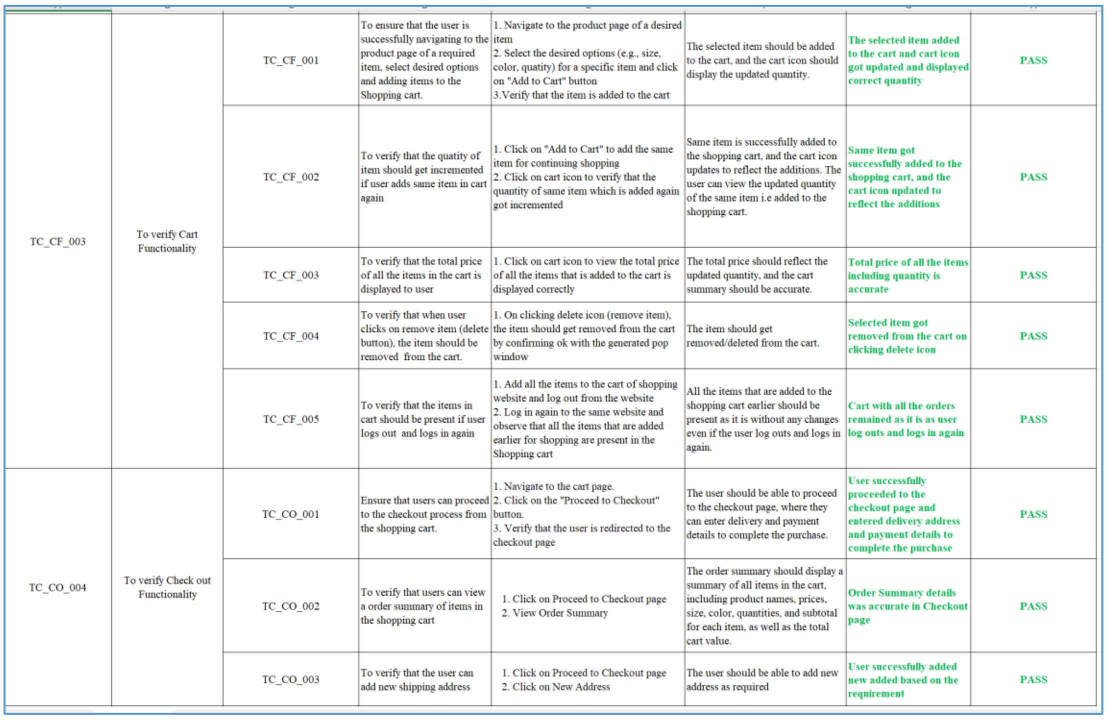
Creating manual cases and automation test scripts is a key part of trying out to systematically become aware of software defects. To carry out Manual Testing, Test cases for a buying demo website are carefully designed to validate person registration, person login, purchasing cart, and checkout technique functionality. Each test case follows a established layout to ensure thorough trying out of practical requirements, with specific identifiers and a part of large test scenarios.

Manual Test Case Design:



Test Scenario ID	Test Case	Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Test Status
TC_URF_001	To verify the User Registration functionality of a Shopping Webpage	TC_URF_Registration_001	To ensure that the new user is able to register successfully with valid information on a Shopping website.	1. Open the browser 2. Enter the URL within the browser 3. Click on "Create an Account" link in the header of the homepage 4. Fill in valid information in all required fields 5. Click on the "Register" button	User should get registered successfully	New User got registered successfully	PASS
		TC_URF_Registration_002	Attempting to register with an already existing email address to ensure that the system does not allow registration with an email address that is already associated with an existing user account.	1. Open the browser 2. Enter the URL within the browser 3. Click on "Create an Account" link in the header of the homepage 4. Enter an email address that is already registered with an existing user account on the website 5. Click on the "Register" or "Sign Up" button to submit the registration form	User should not get registered and shoud display error message "There is already an account with this email address"	Existing user failed to register with the already existing email address	PASS
TC_ULF_002	To verify the User Login functionality of a Shopping website	TC_ULF_Login_001	Enter a Valid Email & Valid Password	1. Open the browser 2. Enter the URL within the browser 3. Click on "Sign in" link in the header of the homepage 4. Enter Valid Email & Valid Password 5. Click on the "Sign in" button	User should login successfully and redirect to the homepage	User logged in successfully	PASS
		TC_ULF_Login_002	Enter a Valid Email & Invalid Password	1. Enter Valid Email & Invalid Password 2. Click on the "Sign in" button	An error message should be displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	An Error message got displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	PASS
		TC_ULF_Login_003	Enter a Invalid Email & Valid Password	1. Enter Invalid Email & Valid Password 2. Click on the "Sign in" button	An error message should be displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	An Error message got displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	PASS
		TC_ULF_Login_004	Enter a Invalid Email & Invalid Password	1. Enter Invalid Email & Invalid Password 2. Click on the "Sign in" button	An error message should be displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	An Error message got displayed as "The account sign-in was incorrect or your account is disabled temporarily. Please wait and try again later."	PASS

Figure 17: Manual Test Case 1
[**Source:** This thesis specific diagram was developed by the author.]

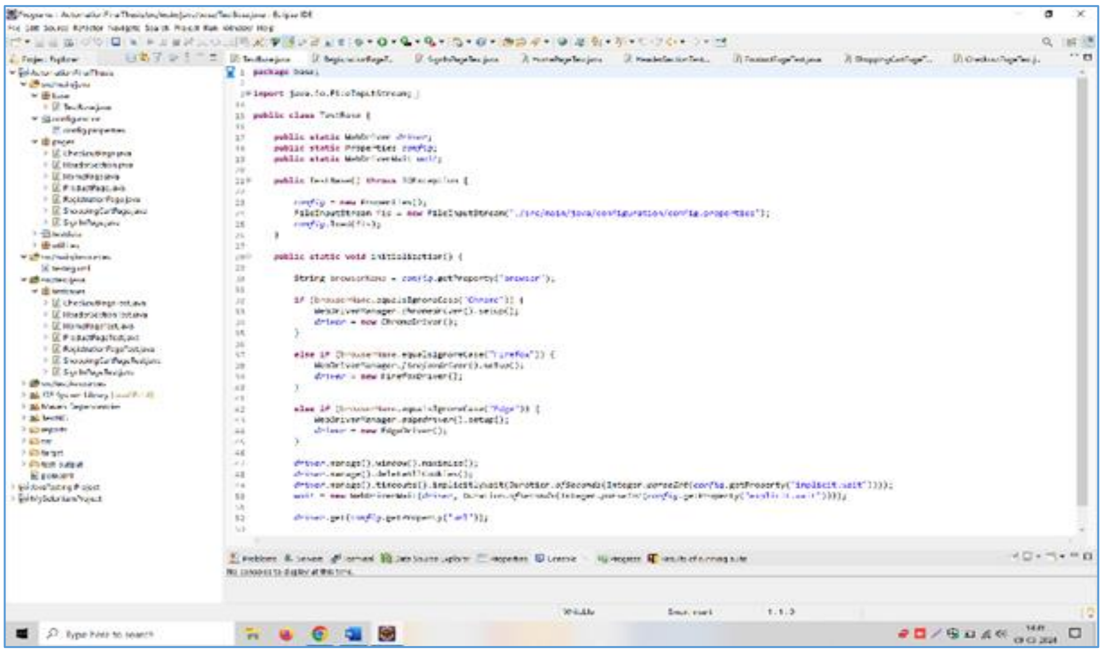


TC_CF_003	To verify Cart Functionality	TC_CF_001	To ensure that the user is successfully navigating to the product page of a required item, select desired options and adding items to the Shopping cart.	1. Navigate to the product page of a desired item 2. Select the desired options (e.g., size, color, quatity) for a specific item and click on "Add to Cart" button 3.Verify that the item is added to the cart	The selected item should be added to the cart, and the cart icon should display the updated quantity.	**The selected item added to the cart and cart icon got updated and displayed correct quantity**	**PASS**
		TC_CF_002	To verify that the quatity of item should get incremented if user adds same item in cart again	1. Click on "Add to Cart" to add the same item for continuing shopping 2. Click on cart icon to verify that the quantity of same item which is added again got incremented	Same item is successfully added to the shopping cart, and the cart icon updates to reflect the additions. The user can view the updated quantity of the same item i.e added to the shopping cart.	**Same item got successfully added to the shopping cart, and the cart icon updated to reflect the additions**	**PASS**
		TC_CF_003	To verify that the total price of all the items in the cart is displayed to user	1. Click on cart icon to view the total price of all the items that is added to the cart is displayed correctly	The total price should reflect the updated quantity, and the cart summary should be accurate.	**Total price of all the items including quantity is accurate**	**PASS**
		TC_CF_004	To verify that when user clicks on remove item (delete button), the item should be removed from the cart.	1. On clicking delete icon (remove item), the item should get removed from the cart by confirming ok with the generated pop window	The item should get removed/deleted from the cart.	**Selected item got removed from the cart on clicking delete icon**	**PASS**
		TC_CF_005	To verify that the items in cart should be present if user logs out and logs in again	1. Add all the items to the cart of shopping website and log out from the website 2. Log in again to the same website and observe that all the items that are added earlier for shopping are present in the Shopping cart	All the items that are added to the shopping cart earlier should be present as it is without any changes even if the user log outs and logs in again.	**Cart with all the orders remained as it is as user log outs and logs in again**	**PASS**
TC_CO_004	To verify Check out Functionality	TC_CO_001	Ensure that users can proceed to the checkout process from the shopping cart.	1. Navigate to the cart page. 2. Click on the "Proceed to Checkout" button. 3. Verify that the user is redirected to the checkout page	The user should be able to proceed to the checkout page, where they can enter delivery and payment details to complete the purchase.	**User successfully proceeded to the checkout page and entered delivery address and payment details to complete the purchase**	**PASS**
		TC_CO_002	To verify that users can view a order summary of items in the shopping cart	1. Click on Proceed to Checkout page 2. View Order Summary	The order summary should display a summary of all items in the cart, including product names, prices, size, color, quantities, and subtotal for each item, as well as the total cart value.	**Order Summary details was accurate in Checkout page**	**PASS**
		TC_CO_003	To verify that the user can add new shipping address	1. Click on Proceed to Checkout page 2. Click on New Address	The user should be able to add new address as required	**User successfully added new added based on the requirement**	**PASS**

Figure 18: Manual Test Case 2
[**Source:** This thesis specific diagram was developed by the author.]

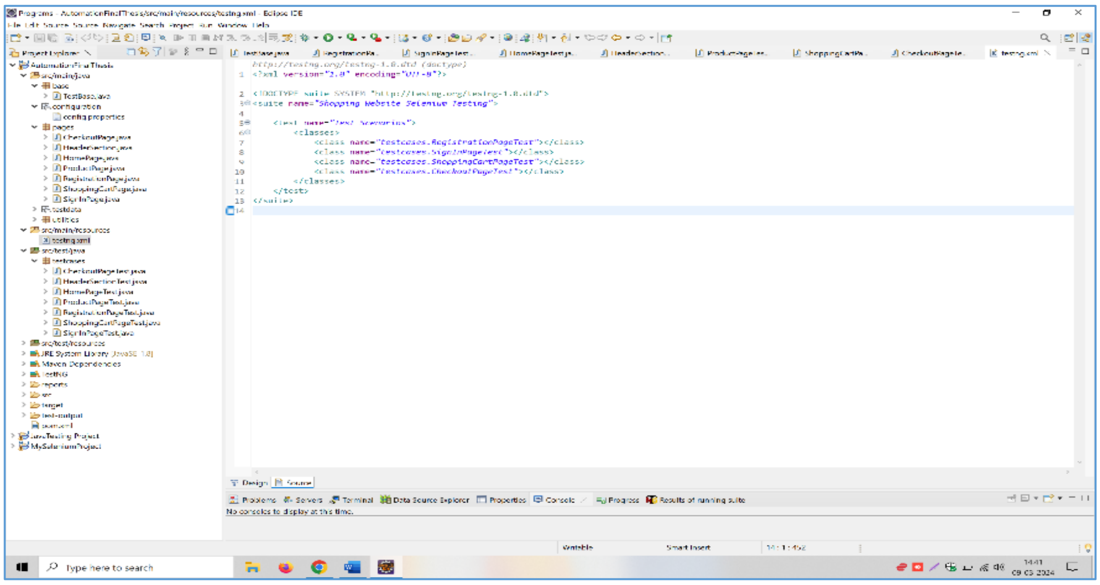
Important Components of Manual Test Case Design illustrated in the Figure: Test Scenario ID serves as a high-level categorization for related test cases, like ‘TC_URF_001’ for user registration. Each test case has a unique Test Case ID for tracking and managing purposes, e.g., ‘TC_URF_Registration_001’. Test Case Description outlines what the test case aims to validate, like user registration or email duplication prevention. Test Steps provide a detailed procedure for testers to follow, including browser opening and data input. Expected Result states the anticipated outcome if the application functions correctly, while Actual Result reveals the outcome of the test execution. Test Status indicates whether the test case passed or failed, aligning with the expected result. The manual test cases are meticulously crafted for precision and repeatability, with clear expected results to determine test success. They cover both positive scenarios like successful user registration and negative scenarios like existing email registrations or incorrect logins. The attention to detail in steps and outcomes ensures thorough testing of nuanced application behaviour’s including cart functionality persistence between user sessions. Overall, the detailed manual test case design showcased in the screenshots highlights careful planning, documentation, and coverage of functional paths for robust software validation.

Automation Test Scripts Design:



```java
package base;

import java.io.FileInputStream;

public class TestBase {

    public static WebDriver driver;
    public static Properties config;
    public static WebDriverWait wait;

    public TestBase() throws IOException {

        config = new Properties();
        FileInputStream fis = new FileInputStream("./src/main/java/configuration/config.properties");
        config.load(fis);
    }

    public static void initialization() {

        String browserName = config.getProperty("browser");

        if (browserName.equalsIgnoreCase("Chrome")) {
            WebDriverManager.chromedriver().setup();
            driver = new ChromeDriver();
        }

        else if (browserName.equalsIgnoreCase("Firefox")) {
            WebDriverManager.firefoxdriver().setup();
            driver = new FirefoxDriver();
        }

        else if (browserName.equalsIgnoreCase("Edge")) {
            WebDriverManager.edgedriver().setup();
            driver = new EdgeDriver();
        }

        driver.manage().window().maximize();
        driver.manage().deleteAllCookies();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(Integer.parseInt(config.getProperty("implicit.wait"))));
        wait = new WebDriverWait(driver, Duration.ofSeconds(Integer.parseInt(config.getProperty("explicit.wait"))));

        driver.get(config.getProperty("url"));
```

Figure 19: Automation Script for TestBase.
[**Source:** This thesis specific diagram was developed by the author.]



```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Shopping Website Selenium Testing">

    <test name="Test Scenarios">
        <classes>
            <class name="testcases.RegistrationPageTest"></class>
            <class name="testcases.SignInPageTest"></class>
            <class name="testcases.ShoppingCartPageTest"></class>
            <class name="testcases.CheckoutPageTest"></class>
        </classes>
    </test>
</suite>
```

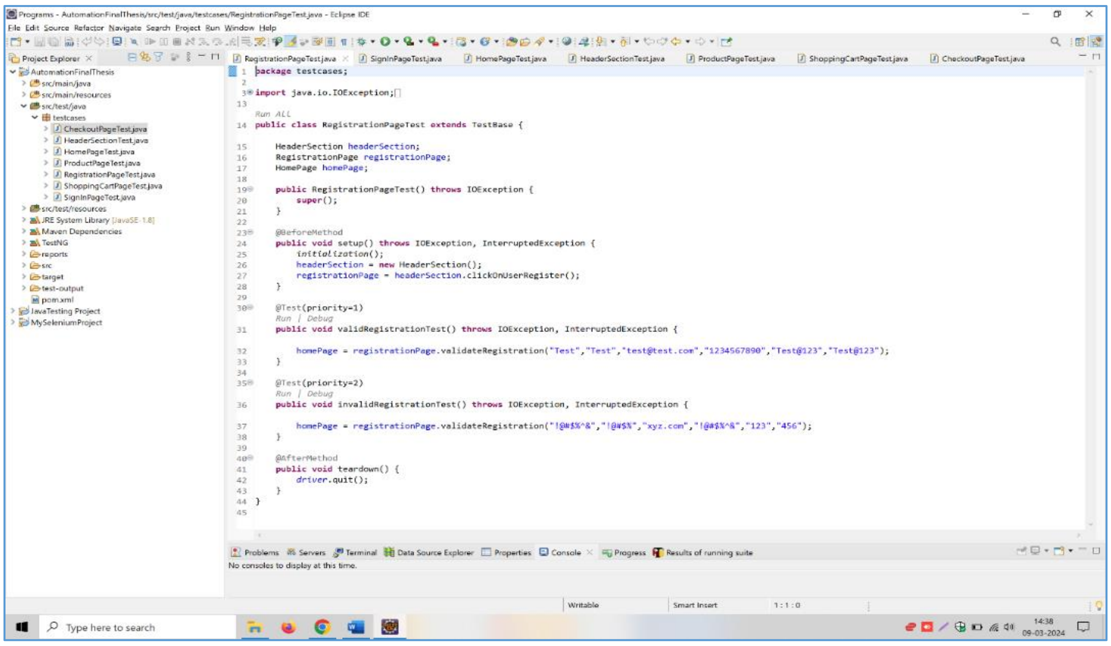
Figure 20: TestNG.xml file.
[**Source:** This thesis specific diagram was developed by the author.]

The evaluation of the different elements of the framework consists of following sections:

- **Page Classes (pages package):** These classes represent individual pages of the web application, with methods corresponding to the functionalities provided by these pages. For instance, Home Page, Sign-In-Page, Registration Page, Product Page, Shopping-Cart-Page, and Checkout Page contain methods that interact with the elements on these pages.
- **Test Classes (testcases package):** Each test class corresponds to a page class and contains test methods annotated with JUnit annotations (@Test). The methods in these classes call the page methods to perform actions and assertions to verify the application's functionality. For example, Registration-Page-Test contains tests for the registration functionality.
- **TestBase Class:** This class serves as the base class for all test classes. It contains common setup and teardown methods that initialize and clean up the test environment before and after each test. It manages the WebDriver instance, loads properties from a configuration file, and sets up wait conditions.
- **Configuration (configuration package):** There's a config.properties file that stores configurable parameters like browser type, URL, and credentials. The TestBase class loads these properties to be used throughout the tests.
- **TestNG XML (testng.xml):** This XML file configures the test run, specifying which test classes to execute. It enables batch running of tests and allows for easy integration with CI/CD pipelines.
- **WebDriver Initialization:** Depending on the browser specified in the config.properties file, the appropriate WebDriver is initialized. The browsers listed are Chrome, Firefox, and Edge.
- **Test Methods:** The test methods use the Page Object methods to perform actions on the web application and validate the outcomes using assertions. There are examples of both positive and negative test cases.

This framework also includes annotations like @BeforeMethod and @AfterMethod for setup and teardown routines, @Test for test methods, and @Test(priority=x) to sequence the tests. The use of throws IO Exception indicates that exception handling is in place for IO-related operations.

To conduct automated testing on a Shopping Demo website, programming scripts are developed using a language that is compatible with Selenium, such as Java, as part of a Selenium-based automation framework. These scripts utilize the Page Object Model (POM) design pattern for web applications enhances test maintenance and reduces code duplication, enabling them to communicate with web browsers and carry out automated testing tasks via the Selenium WebDriver API. The scripts are structured within a Maven project setup and leverage the TestNG framework to control the testing process. Scripts are crafted using TestNG, which is a sophisticated testing framework featuring enhanced annotations and organization of test methods. It also facilitates data-driven testing and integrates with Maven to handle dependencies and execute tests during the build cycle.



```java
package testcases;

import java.io.IOException;

public class RegistrationPageTest extends TestBase {

    HeaderSection headerSection;
    RegistrationPage registrationPage;
    HomePage homePage;

    public RegistrationPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        registrationPage = headerSection.clickOnUserRegister();
    }

    @Test(priority=1)
    public void validRegistrationTest() throws IOException, InterruptedException {

        homePage = registrationPage.validateRegistration("Test","Test","test@test.com","1234567890","Test@123","Test@123");
    }

    @Test(priority=2)
    public void invalidRegistrationTest() throws IOException, InterruptedException {

        homePage = registrationPage.validateRegistration("!@#$%^&","!@#$%","xyz.com","!@#$%^&","123","456");
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

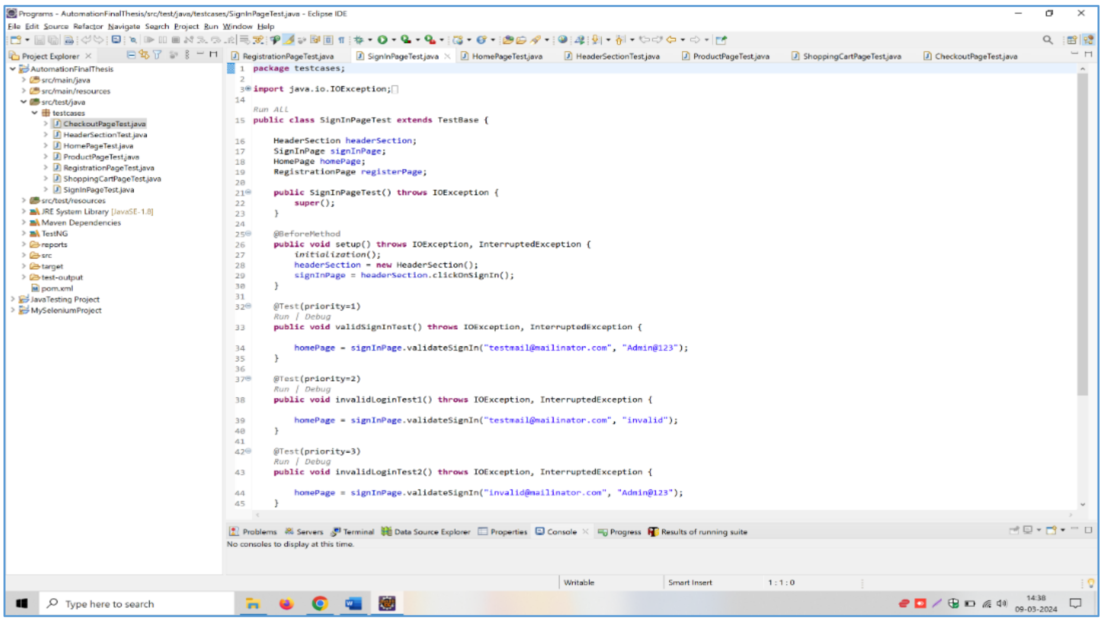
No consoles to display at this time.

Figure 21: Automation Script for Registration page.
[**Source:** This thesis specific diagram was developed by the author.]

Registration Page Test

This test class focuses on the functionality of user registration. It includes tests for validating successful registration with correct details and handling invalid registration attempts.

The RegistrationPageTest class ensures that the registration process on the website functions correctly. It includes positive and negative test scenarios to verify that only users with valid details can register and that appropriate messages are displayed when incorrect details are provided.



```java
package testcases;

import java.io.IOException;

public class SignInPageTest extends TestBase {

    HeaderSection headerSection;
    SignInPage signInPage;
    HomePage homePage;
    RegistrationPage registerPage;

    public SignInPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        signInPage = headerSection.clickOnSignIn();
    }

    @Test(priority=1)
    public void validSignInTest() throws IOException, InterruptedException {

        homePage = signInPage.validateSignIn("testmail@mailinator.com", "Admin@123");
    }

    @Test(priority=2)
    public void invalidLoginTest1() throws IOException, InterruptedException {

        homePage = signInPage.validateSignIn("testmail@mailinator.com", "invalid");
    }

    @Test(priority=3)
    public void invalidLoginTest2() throws IOException, InterruptedException {

        homePage = signInPage.validateSignIn("invalid@mailinator.com", "Admin@123");
    }
```

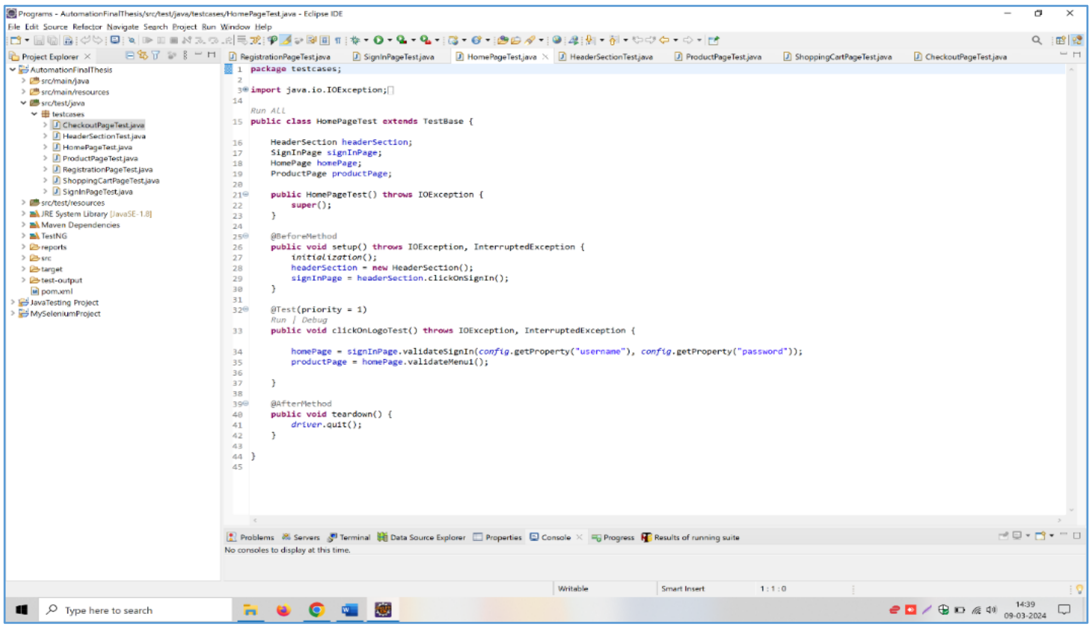
No consoles to display at this time.

Figure 22: Automation Script for Sign-in page.
[**Source:** This thesis specific diagram was developed by the author.]

Sign In Page Test

The SignInPageTest tests the login functionality. It contains methods to check if a user can log in with valid credentials and tests for login attempts with invalid credentials.

SignInPageTest validates the sign-in process of the application. It ensures that users with valid credentials can access the system and that access is denied to users with incorrect credentials, thus safeguarding against unauthorized access.



```java
package testcases;

import java.io.IOException;

Run All
public class HomePageTest extends TestBase {

    HeaderSection headerSection;
    SignInPage signInPage;
    HomePage homePage;
    ProductPage productPage;

    public HomePageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        signInPage = headerSection.clickOnSignIn();
    }

    @Test(priority = 1)
    Run | Debug
    public void clickOnLogoTest() throws IOException, InterruptedException {

        homePage = signInPage.validateSignIn(config.getProperty("username"), config.getProperty("password"));
        productPage = homePage.validateMenu1();

    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }

}
```

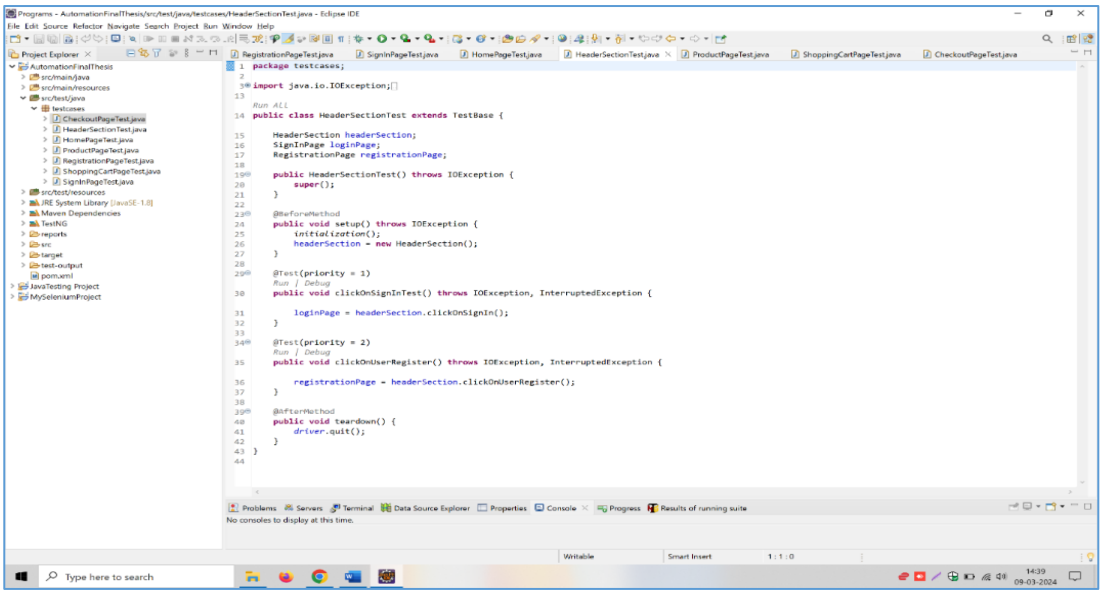
```
No consoles to display at this time.
```

Figure 23: Automation Script for Home page.
[**Source:** This thesis specific diagram was developed by the author.]

Home Page Test

This class tests features available on the homepage, such as navigation and logging out. One of the tests checks if the homepage is accessible after a successful login.

The HomePageTest class confirms the homepage’s integrity and navigability post-login. It checks the functionality of core elements, ensuring the user experience remains consistent upon accessing the homepage.



```java
package testcases;

import java.io.IOException;

public class HeaderSectionTest extends TestBase {

    HeaderSection headerSection;
    SignInPage loginPage;
    RegistrationPage registrationPage;

    public HeaderSectionTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException {
        initialization();
        headerSection = new HeaderSection();
    }

    @Test(priority = 1)
    public void clickOnSignInTest() throws IOException, InterruptedException {

        loginPage = headerSection.clickOnSignIn();
    }

    @Test(priority = 2)
    public void clickOnUserRegister() throws IOException, InterruptedException {

        registrationPage = headerSection.clickOnUserRegister();
    }

    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

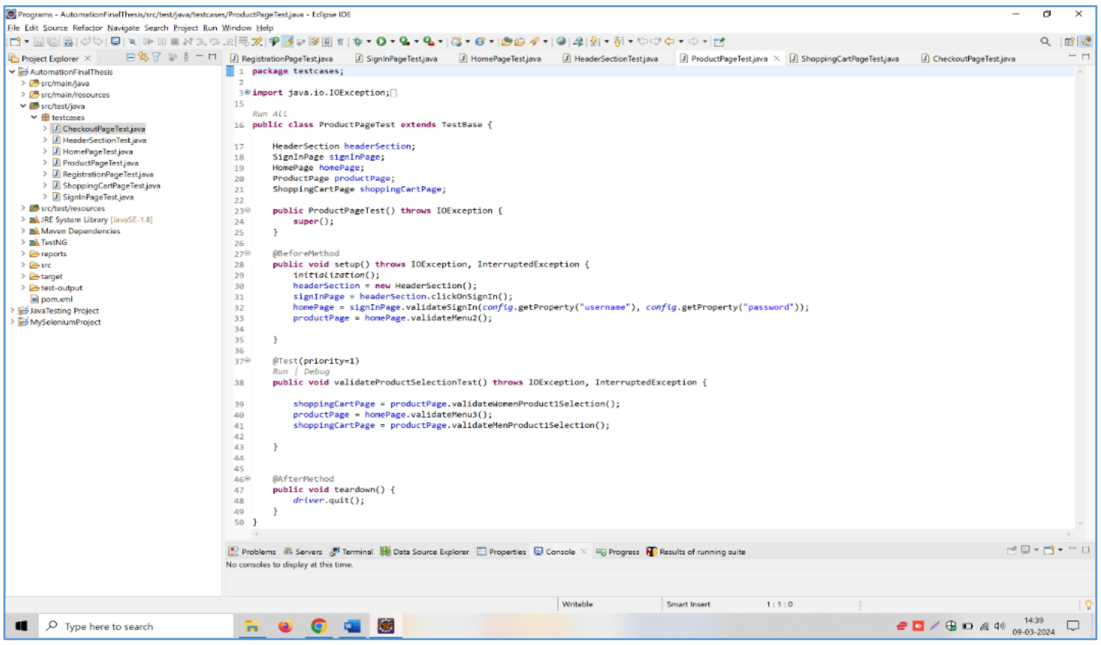
No consoles to display at this time.

Figure 24: Automation script for Header-section page.
[**Source:** This thesis specific diagram was developed by the author.]

Header Section Test

In the HeaderSectionTest, the tests focus on the header section of the website, checking functionalities like navigating to the sign-in and registration pages from the header.

HeaderSectionTest is designed to confirm that the header section of the website provides the necessary navigation functionalities, allowing users to move to the sign-in and registration pages seamlessly.



```java
package testcases;

import java.io.IOException;

public class ProductPageTest extends TestBase {

    HeaderSection headerSection;
    SignInPage signInPage;
    HomePage homePage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;

    public ProductPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        signInPage = headerSection.clickOnSignIn();
        homePage = signInPage.validateSignIn(config.getProperty("username"), config.getProperty("password"));
        productPage = homePage.validateMenu2();

    }

    @Test(priority=1)
    public void validateProductSelectionTest() throws IOException, InterruptedException {

        shoppingCartPage = productPage.validateWomenProduct1Selection();
        productPage = homePage.validateMenu3();
        shoppingCartPage = productPage.validateMenProduct1Selection();

    }


    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

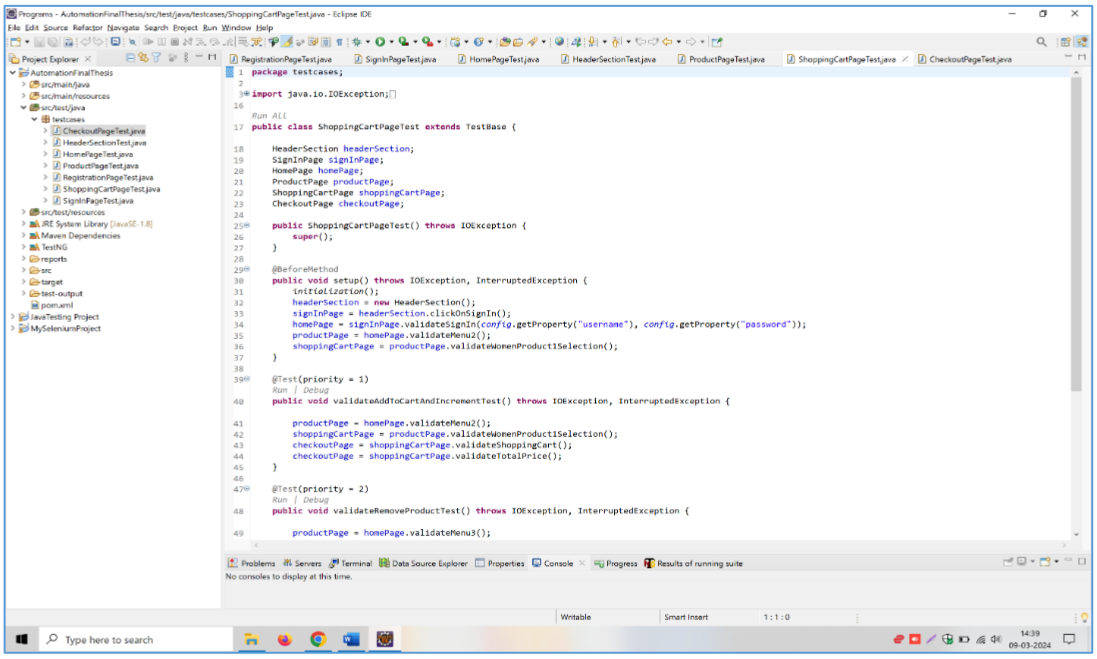
```
No consoles to display at this time.
```

Figure 25: Automation script for Product page.
[**Source:** This thesis specific diagram was developed by the author.]

Product Page Test

Tests within the ProductPageTest class verify the behavior of the product page, such as selecting items and navigating to the shopping cart.

The ProductPageTest evaluates the product selection process, ensuring that users can successfully add items to their shopping cart from the product page, enhancing the shopping experience.



```
package testcases;

import java.io.IOException;

public class ShoppingCartPageTest extends TestBase {

    HeaderSection headerSection;
    SignInPage signInPage;
    HomePage homePage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;
    CheckoutPage checkoutPage;

    public ShoppingCartPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        signInPage = headerSection.clickOnSignIn();
        homePage = signInPage.validateSignIn(config.getProperty("username"), config.getProperty("password"));
        productPage = homePage.validateMenu2();
        shoppingCartPage = productPage.validateWomenProduct1Selection();
    }

    @Test(priority = 1)
    public void validateAddToCartAndIncrementTest() throws IOException, InterruptedException {

        productPage = homePage.validateMenu2();
        shoppingCartPage = productPage.validateWomenProduct1Selection();
        checkoutPage = shoppingCartPage.validateShoppingCart();
        checkoutPage = shoppingCartPage.validateTotalPrice();
    }

    @Test(priority = 2)
    public void validateRemoveProductTest() throws IOException, InterruptedException {

        productPage = homePage.validateMenu3();
```

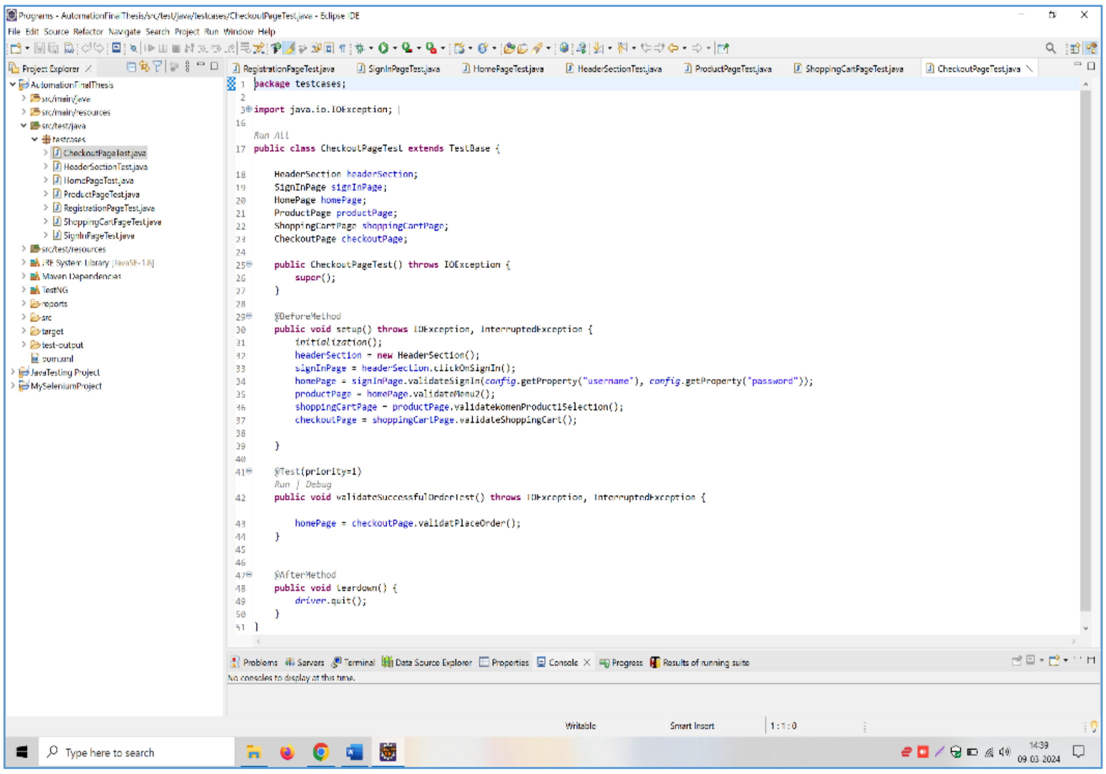
No consoles to display at this time.

Figure 26: Automation script for Shopping-cart page.
[**Source:** This thesis specific diagram was developed by the author.]

Shopping Cart Page Test

This class focuses on the shopping cart's functionalities, including adding items, incrementing item quantities, and removing products from the cart.

ShoppingCartPageTest assures that the shopping cart operates as intended, with tests to validate the addition and removal of products as well as the updating of product quantities within the cart.



```java
package testcases;

import java.io.IOException;

public class CheckoutPageTest extends TestBase {

    HeaderSection headerSection;
    SignInPage signInPage;
    HomePage homePage;
    ProductPage productPage;
    ShoppingCartPage shoppingCartPage;
    CheckoutPage checkoutPage;

    public CheckoutPageTest() throws IOException {
        super();
    }

    @BeforeMethod
    public void setup() throws IOException, InterruptedException {
        initialization();
        headerSection = new HeaderSection();
        signInPage = headerSection.clickOnSignIn();
        homePage = signInPage.validateSignIn(config.getProperty("username"), config.getProperty("password"));
        productPage = homePage.validateMenu2();
        shoppingCartPage = productPage.validateWomenProduct1Selection();
        checkoutPage = shoppingCartPage.validateShoppingCart();

    }

    @Test(priority=1)
    public void validateSuccessfulOrderTest() throws IOException, InterruptedException {

        homePage = checkoutPage.validatPlaceOrder();
    }


    @AfterMethod
    public void teardown() {
        driver.quit();
    }
}
```

Figure 27: Automation script for Check-out page.
[**Source:** This thesis specific diagram was developed by the author.]

Checkout Page Test

The CheckoutPageTest ensures that the checkout process, including order placements, is functioning correctly. It tests for a successful order completion after the user has added items to the cart.

CheckoutPageTest is key for verifying the checkout process, confirming that users can place orders successfully after adding their desired products to the shopping cart.

Each test class aligns with a specific section or functionality of the web application. They all inherit from the TestBase class, which handles common setup and teardown tasks. The classes contain @Test methods, which are actual test cases, and use the Page Object Model to interact with the web application under test. This structured approach ensures that the tests are maintainable and that the application's functionalities are thoroughly validated.

4.5.3 Challenges in Test Case Maintenance

Maintaining test cases is vital but difficult in the software testing process. It entails modifying them to match the application, environment, or requirements. Many challenges are faced during test case maintenance which are listed below:

- As software applications evolve with new features and modifications, test cases must be continuously updated to align with these changes. This can be a time-consuming process, particularly for complex applications with frequent releases.
- Tests that are unreliable, resulting in varying outcomes, can be a significant challenge to maintain. Identifying if a failure is caused by a flaw in the application or an issue with the test itself requires thorough investigation and substantial effort.
- Maintaining the significance and authenticity of test data in complex systems is challenging because of the complex data dependencies that exist.
- In automated testing, changes in the UI or API can cause automated tests to break, necessitating frequent updates to the test scripts. This fragility can lead to a high maintenance overhead.
- Updates or changes in testing tools, browsers, operating systems, or other components of the testing environment can lead to test failures, requiring tests to be reviewed and updated accordingly.
- The need for specialized knowledge to maintain certain test cases can pose a challenge, especially when team members with the necessary expertise are unavailable.
- Without adequate traceability to requirements, managing test cases can become aimless, making it difficult to prioritize updates and ensure that the test suite stays focused on validating the critical functionalities of the application.
- Poor documentation of test cases complicates maintenance efforts, as it requires testers to spend additional time understanding the purpose and steps of the test before they can make updates.
- Limited resources, in terms of both time and personnel, can lead to a backlog of maintenance tasks, putting pressure on the testing team and potentially compromising the quality of testing.

4.6 Real-world Case Studies Implementation

4.6.1 Case Study Selection Criteria

When selecting real-world case studies for the implementation of Selenium WebDriver automation scripts, several key criteria were taken into account to ensure a comprehensive evaluation of the framework's capabilities:

- **Web Application Complexity:** Case studies were chosen based on the complexity of the web applications, including multi-page workflows, dynamic content, and responsive design elements. This criterion ensures that the framework is tested against applications that simulate a realistic user interaction scenario.
- **Variety of Web Elements:** Applications with a wide variety of web elements such as forms, dropdowns, modals, and pop-ups were selected. This diversity tests the robustness of the locator strategies and interaction methods defined in the page objects.
- **Functionality Coverage:** The selected applications cover a range of functionalities, from user registration and sign-in processes to product selection and shopping cart management. This allows for an exhaustive assessment of the test scripts across different functional domains.
- **User Interaction Flows**: Applications with intricate user interaction flows, including both linear and non-linear navigation paths, were considered. These flows help validate the framework's ability to handle complex user journeys and state management.
- **Error Handling and Negative Testing:** Case studies that provide ample scenarios for negative testing and error handling were preferred. The ability to gracefully handle unexpected scenarios and recover from errors is a crucial aspect of automated testing.
- **Cross-Browser Testing:** The need for cross-browser compatibility in the modern web landscape necessitated the inclusion of case studies that can be tested across multiple browsers, ensuring the scripts' effectiveness in diverse environments.

4.6.2 Implementation Insights

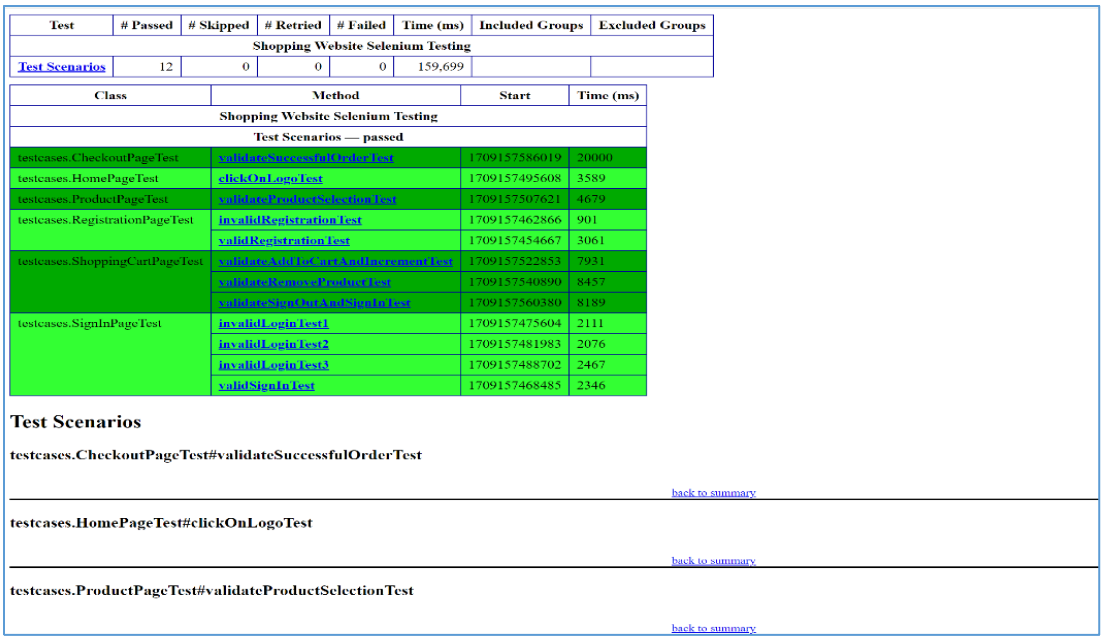
During the implementation of the Selenium WebDriver test scripts, several insights were gathered that inform best practices and optimization strategies for automated testing frameworks:

- **Page Object Model Effectiveness:** The implementation reinforced the effectiveness of the Page Object Model in enhancing test maintainability and readability, as evidenced by the structured and modular codebase of the page classes.
- **Configurable Test Data:** The utilization of a config.properties file demonstrated the value of keeping test data and configurations separate from the script logic. This separation allows for easy adjustments without modifying the core test scripts.
- **Robust Locator Strategies:** Implementing the tests underscored the importance of robust locator strategies. The ability to identify web elements reliably under different conditions was critical for the stability of the test execution.
- **Asynchronous Behavior Handling:** Handling the asynchronous behavior of web applications through explicit and implicit waits was a key learning point, ensuring that the tests are stable and less flaky.
- **Negative Test Scenarios:** Incorporating negative test scenarios proved to be vital in ensuring the application’s resilience against invalid inputs and unexpected user behavior.
- **Continuous Integration Readiness:** The framework’s compatibility with TestNG and the creation of a testng.xml configuration file facilitated the integration of the test suite into CI/CD pipelines, allowing for automated triggers of test runs.
- **Scalability and Flexibility:** The scalability and flexibility of the framework were tested by progressively adding more complex test scenarios and ensuring the test infrastructure can handle the increased load.
- **Cross-Browser Testing:** Real-world implementation provided insights into cross-browser issues and reinforced the need for comprehensive cross-browser testing to ensure consistent user experiences.

The diploma thesis can delve deeper into each of those insights, providing detailed analysis and examples from the case research to illustrate the real-world applicability and demanding situations of Selenium WebDriver automated testing.

4.6.3 Practical Outcomes

Manual And Automation Test Execution:



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Shopping Website Selenium Testing							
Test Scenarios	12	0	0	0	159,699		

Class	Method	Start	Time (ms)
Shopping Website Selenium Testing			
Test Scenarios — passed			
testcases.CheckoutPageTest	validateSuccessfulOrderTest	1709157586019	20000
testcases.HomePageTest	clickOnLogoTest	1709157495608	3589
testcases.ProductPageTest	validateProductSelectionTest	1709157507621	4679
testcases.RegistrationPageTest	invalidRegistrationTest	1709157462866	901
	validRegistrationTest	1709157454667	3061
testcases.ShoppingCartPageTest	validateAddToCartAndIncrementTest	1709157522853	7931
	validateRemoveProductTest	1709157540890	8457
	validateSignOutAndSignInTest	1709157560380	8189
testcases.SignInPageTest	invalidLoginTest1	1709157475604	2111
	invalidLoginTest2	1709157481983	2076
	invalidLoginTest3	1709157488702	2467
	validSignInTest	1709157468485	2346

Test Scenarios

testcases.CheckoutPageTest#validateSuccessfulOrderTest

back to summary

testcases.HomePageTest#clickOnLogoTest

back to summary

testcases.ProductPageTest#validateProductSelectionTest

back to summary

Figure 28: Automation Test Execution report.
[**Source:** This thesis specific diagram was developed by the author.]

Comparison of Time taken during Test execution using Manual & Automation testing.

Sl.No	Test Cases	Time Taken in Manual Testing	Time Taken in Automation testing
1	Registering a New User	30 seconds	3 seconds
2	Registering an Existing User	30 seconds	1 second
3	User login with Valid email & Valid password	20 seconds	2.3 seconds
4	User login with Valid email & Invalid password	20 seconds	2.1 seconds
5	User login with Invalid email & Valid password	20 seconds	2 seconds
6	User login with Invalid email & Invalid password	25 seconds	2.4 seconds
7	Successfully adding desired product to the cart	60 seconds	15 seconds
8	Adding same item & cart getting incremented	60 seconds	
9	Total price of all the items in the cart is displayed	150 seconds	60 seconds
10	On clicking remove item, the item should be removed from the cart.	150 seconds	
11	Items in cart should be present if user logs out and logs in again	120 seconds	50 seconds
12	Users can proceed to the checkout process from the shopping cart	5 minutes	20 seconds
13	Users can view an order summary of items in the shopping cart	5 minutes	
14	User can add new shipping address	7 minutes	NA
TOTAL TIME TAKEN		**28 Minutes 40 Seconds**	**2 Minutes 60 Seconds**

Table 16: Comparison of Time Taken.
[**Source:** This thesis specific table was developed by the author.]

The TestNG report shown in Figure. 28 depicts automated test execution report for a shopping demo website where functional testing was executed using both manual and automated test scripts. The report outlines the execution of test cases related to user registration, login, cart functionality, and checkout processes. This includes both positive and negative scenarios to ensure comprehensive testing of the application’s functionality. And Table. 16 shows the comparision of time taken to executed similar test cases using both manual and automated testing approaches.

Based on the TestNG report, we can infer the following:

- **User Registration Functionality:** The test cases includes scenarios such as registering with valid details (positive test) and attempting to register with already existing email addresses (negative test). The automated scripts executed by TestNG have validated the application’s response to each of these inputs, such as successful account creation or appropriate error messages.

- **User Login Functionality:** This involved automated tests for valid login credentials (positive test) and invalid login attempts (negative tests). The TestNG report indicates the time taken for each test, reflecting the efficiency of automated checks against these criteria.

- **Cart Functionality:** Automated test cases for the shopping cart have included adding items to the cart, updating item quantities, and persisting cart items when a user logs out and back in. Again, the report provides execution times, showing how quickly the automation can verify cart behaviors.

- **Checkout Functionality:** The automated checkout process testing involve successful order placement (positive scenario) and order placement with invalid details (negative scenario). The TestNG framework have executed these tests to validate the workflow and report any discrepancies from expected outcomes.

- Comparing the execution times for manual versus automated tests can provide insights into performance and efficiency. Manual testing, while crucial for exploratory and usability checks, typically takes longer due to the need for human interaction. Automated tests, however, can be executed quickly and repeatedly with TestNG, which is evident from the timestamps and durations specified in the report. Automated tests also offer the advantage of running at off-peak times or concurrently, thereby not adding to the time constraints of a sprint or development cycle.

- For instance, an automated test that checks for valid user registration will consistently enter the same data and expect the same result, which can be completed in a matter of seconds as per the TestNG report timestamps. In contrast, a manual tester may take a few minutes to complete the same task, and the time vary slightly with each execution.

- The report also shows longer execution times for more complex scenarios, such as those involving multiple steps or validations. These findings highlight the effectiveness of automation in performing mundane, repetitive, and data-intensive tests, allowing manual testers to focus on more intricate tests that require human insight.

In summary, the TestNG report for the demo shopping website provides valuable data on the time efficiency of automated testing compared to manual testing. While both methods have their place in a balanced testing strategy, the synergy of both is leveraged to ensure thorough quality assurance, with automation significantly enhancing test efficiency and performance.

4.7 Best Practices Integration

4.7.1 Synergies Between Manual and Automation Testing

Combining manual and automated testing approaches can lead to substantial synergies that enhance the general efficiency and effectiveness of software testing, these synergies are built on the unique capabilities of both methods and when properly utilized result in a tough and quick system of testing here are some of key benefits realized by integrating both manual and automated tests.

- **Increased Test Coverage:** Automation can handle repetitive and high-volume test cases, allowing manual testers to focus on exploratory, usability, and ad-hoc testing. This distribution of work increases overall test coverage—automation ensures consistency in regression testing, while manual testing explores new or complex user scenarios.

- **Efficiency and Speed:** Automated tests can quickly perform numerous tasks that take much longer if done manually, leading to faster test cycles. Selective manual testing can be utilized to specifically target areas that necessitate human judgment, such as visual appraisals and evaluations of user experience.

- **Resource Optimization:** By employing automation for repetitive and regression tests, organizations can make better use of human resources, allocating skilled testers to areas where their expertise is most needed, such as in test planning, test case design, and high-level test execution.

- **Improved Accuracy and Reduced Human Error:** Automated testing provides precise and consistent execution of predefined test cases, reducing the risk of human error associated with repetitive manual testing tasks.

- **Feedback Loop Enhancement:** Automated tests can be integrated into the Continuous Integration/Continuous Deployment (CI/CD) pipeline, providing immediate feedback to developers. Manual testing can complement this by providing more in-depth understanding of possible problems or enhancements, thereby improving the feedback loop for development teams.

- **Cost-Effectiveness:** Over time, automation can lead to cost savings by reducing the time and resources required for repetitive testing. Manual testing can then be strategically employed for tasks that are not cost-effective to automate.

- **Skill Development:** Cross-training team members in both manual and automated testing can lead to a more versatile and skilled testing workforce capable of tackling various testing challenges.

- **Quality Assurance:** Combining manual and automated testing can lead to higher quality software, as different testing methods can catch different types of issues. This comprehensive approach ensures a more thorough examination of the software product.

- **Risk Management:** Manual testing can be used for high-risk areas or new features where automated tests have not yet been developed. Conversely, automation can be employed to continuously test and monitor the more stable parts of the application, ensuring that any new changes do not introduce regressions.

Hence by combining manual and automated testing methods can create a well-rounded strategy that maximizes the advantages of each. This unified approach can enhance software quality, increase efficiency, improve resource management, and strengthen the testing process. Organizations should acknowledge these benefits and incorporate both methods strategically for optimal testing results.

4.7.2 Recommendations for Optimal Efficiency

To optimize software testing efficiency, it is advised to use a balanced approach that combines manual and automated testing strengths. Identify repetitive tasks for automation to save time and minimize errors, especially in regression, data validation, and performance testing. Manual testing should focus on areas requiring human intuition like exploratory and usability testing. Maintain well-documented automated test scripts and invest in continuous integration tools for efficiency and scalability in testing. Creating a detailed test plan with clear objectives and test cases is essential for effective manual testing. Equip testers with the right tools for managing test cases and collaborating with the development team. Cross-train team members in manual and automated testing to ensure versatility and adaptability. Foster collaboration between developers and testers for better defect resolution. Regularly review and update testing strategies to respond to new challenges. Following these recommendations will maintain an efficient and effective testing process and aligns with software development lifecycle needs by adhering to these recommendations in organizations.

4.8 Future Trends in Software Testing Impact

4.8.1 Emerging Trends in Test Automation

Advancements in technology and changes in software development methodologies are driving the reshaping of software testing through emerging trends in test automation. One significant trend is the increased integration of artificial intelligence (AI) and machine learning (ML) into test automation tools, enabling more intelligent decision-making and predictive analytics. Another trend is the adoption of codeless test automation platforms, making test automation more accessible to a broader range of professionals. The rise of Continuous Integration/Continuous Deployment (CI/CD) and DevOps has also had a profound impact on test automation practices, emphasizing shift-left testing and automation to keep pace with fast-paced software deployments. Additionally, there is a focus on testing in more complex environments, supported by the design of test automation tools to cater to a wide range of platforms and devices. Collaboration and integration of test automation tools with other software development lifecycle tools are also becoming increasingly important for better communication and collaboration within development and testing teams. To remain up to date in test automation, testers and organizations need to constantly evolve their tools and practices to fully utilize the potential of new technologies and methodologies driving the field forward.

4.8.2 AI, ML, Agile, and DevOps Impact

The impact of Artificial Intelligence (AI), Machine Learning (ML), Agile methodologies, and DevOps practices on the domain of software testing is profound and multifaceted. AI and ML are revolutionizing test automation by enabling smarter and more adaptive testing processes. These technologies allow for the automatic generation of test cases, prediction of key risk areas, and more efficient management of the vast datasets generated during testing. Machine learning models can analyze historical test data to identify patterns, predict outcomes, and provide insights that lead to more targeted and effective testing strategies.

Agile methodologies have necessitated a more iterative and continuous approach to testing, integrating testing into the early stages of development to identify and resolve issues more quickly. Agile has promoted the development of test automation tools that are flexible and can accommodate rapid changes to the codebase without the need for extensive rework.

DevOps practices further integrate testing into the continuous integration and deployment pipeline, ensuring that automated tests are run frequently and consistently, which helps in identifying defects as soon as they are introduced. The synergy between DevOps and test automation is critical for achieving the rapid deployment cycles that are essential to the DevOps model while maintaining a high standard of software quality.

The combined impact of AI, ML, Agile, and DevOps is creating a more dynamic, responsive, and efficient testing environment. This is characterized by a shift-left in testing practices, where quality assurance is involved at every stage of the software development lifecycle, fostering a culture of continuous improvement, and enabling faster time-to-market for high-quality software products.

5. Results and Discussion

The results of the mixed methods analysis are presented in the section 4.1.2. The study will delve into the quantitative and qualitative data collected to provide a better understanding of automated human testing techniques.

5.1 Analysis of Manual Testing Results

Examining manual testing results from different sizes of projects reveals several patterns. Small projects, which are often characterized by dynamic and agile development environments, shows how manual testing can be flexible when it comes to rapidly changing requirements. When it comes to small projects, manual testing works especially well, when adapted streamlining and frequent changes take precedence. On the other hand, larger projects with complex systems and larger test sets may have trouble with flexible manual testing, potentially sharing the testing process. Intermediate projects between these two extremes finds a balanced approach to manual testing helpful.

Strengths:

Using quantitative data (such as survey and test management tools) and qualitative techniques (such as interviews), the study focuses on the key strengths identified. This can include tasks such as functional testing, exploratory testing, and managing edge cases, which are domains where manual testing shines.

Weaknesses:

Data will be analysed to identify barriers to manual testing, including the need to spend a lot of time repeating activities, the possibility of human error, and potential problems with systems a remarkable in scalability.

Resource management:

Quantitative data will be analysed to understand how manual testing versus automation testing scenarios impacts resource allocation—that is, the number of testers required.

5.2 Analysis of Automation Testing Results

The findings of automation testing show clear advantages for larger projects with more complex test cases that are repeated. Larger projects benefit from increased productivity and reliable results from automated test writing, especially in regression testing and performance testing. While medium-sized businesses benefit from automation, it can be difficult for them to allocate the funds and initial resources needed to automate testing programs small businesses may find it costly in the case of automation outweighs its usefulness when dealing with more compact and dynamic tasks.

Strengths:

The study looks at the benefits of automated testing such as qualitative insights (such as tester experience increasing productivity) and quantitative data (such as faster implementation times). Regression testing, performance testing, and multiple coverage testing are just a few examples of what this can entail.

Weaknesses:

Data will be studied to identify shortcomings of automation testing, including the need for initial investment in tooling and script development, the need for ongoing script maintenance, and the inability to automate testing in any case.

Resource management:

Quantitative data will be analysed to understand how automated testing impacts resource allocation, including reducing the number of manual tests required for specific projects.

5.3 Comparison of Results

Analysis of the results of manual and laboratory tests of various commercial sizes reveals a complex picture. Small businesses that value flexibility and rapid improvement find manual testing appropriate for their goals. Depending on the specifics of the project, medium-sized enterprises combine automated and manual testing in a balanced way. Larger businesses use automation because it is more efficient and scalable. The comparison highlights the importance of tailoring test methods to the specific needs and circumstances of each large-scale project.

Important areas of focus may include:

- **Efficiency:** Check how long it takes to run tests and look for errors at any stage.
- **Effectiveness:** Examine covered test achievement and detection of deficiencies in both human and automated research.
- **Applications:** Evaluate how each strategy affects budget and human resource allocation.
- **Strengths and Weaknesses:** Summarize the main advantages and disadvantages of human and mechanical testing.

5.4 Implications of Findings

The findings of the study highlight the importance of developing assessment strategies that are specific to the size of the organisation. Small businesses benefit from being flexible and investing in manual testing due to their dynamic development environment. To maximize productivity, mid-size companies should deliberately integrate automated manual testing. Large enterprises, with their extensive infrastructure and testing labs, can greatly benefit from a strong automation system. Organizations can use insights to ensure the quality of the software development lifecycle, improve efficiency, and align testing processes with enterprise size.

This section will discuss the broader implications of the survey findings for the software development community based on comparisons and analytical findings. This may include:

- **Selection criteria:** Advice on the best test methodology to use given the nature of the application, project requirements and available resources.
- **A balanced approach:** Emphasize the value of an experimental approach that balances the advantages of manual and automated testing to achieve the best results.
- **Resource Optimization:** Explain how resource allocation decisions between manual and automated testing activities can be made based on the survey data.

5.5 Practical Insights from Case Studies

Real case studies provide useful insights for businesses of all sizes. The flexibility of manual testing helps small businesses startups with limited resources, for example—adjust quickly to changing needs. Software development organizations provide examples of how mid-sized projects use hybrid design, carefully balancing hands-on and automated testing. MNCs, representing large enterprises, demonstrate the scalability and efficiency improvements that can be achieved through extensive automation These case studies provide useful insights that enable companies to make informed decisions through strategies using testing methods to establish their objectives and characteristics.

In this section (as outlined in Section 2.2.4) the research will examine real-world case studies in detail. Examining these case studies can provide the following useful conclusions.

- **Successful implementations:** Examine real-world examples of software development teams that have successfully integrated automated human testing into their implementations.
- **Process and Results:** Calculate the benefits seen in the case studies, such as increased test payment percentages, shorter test schedules, and increased defect detection of the account.
- **Top Techniques:** Identify generalizable best practices from case studies that can be applied to test projects. This may involve developing methods for combining automated and manual testing or for solving specific testing problems.

Specifically, the results and Discussion phase not only analyse the results of human and automated testing, but also change practical and meaningful ways to manage small, medium, and sizes specific needs. Organizations can streamline their testing processes and align testing strategies with company size to ensure a balanced and effective approach to unique software development environments.

6. Conclusion

6.1 Summary of Key Findings

The constant changes and increasing requirements in software development require a careful and sensitive approach to quality assurance. An important part of this effort is software testing, which involves carefully comparing systems against pre-established standards and is necessary to ensure the reliability and performance of applications The testing industry has changed have favoured automation testing, a paradigm of specialized tools and scripts to automatically execute test cases using. Automation testing has replaced manual testing, which was previously controlled by human testers who precise test procedures.

This study examined the advantages and disadvantages of manual and automated testing, highlighting the specific conditions under which each method works best and any underlying limitations. Furthermore, the study examined the various automation frameworks and tools available, analysing their characteristics and limitations. The primary objective was to provide a comprehensive understanding of the software development teams so that they can decide when and how to implement each testing method to ensure maximum coverage and high-quality software delivery.

Getting to know the details: Adapt testing schedules to fit the size of the company

The findings of the study provided the useful realization that there is no one way to choose the best research strategy. The scale of the project greatly influences the efficiency of testing methods.

Small Businesses:

Using hands-on testing to acknowledge agility. Small projects exhibited flexible manual testing, modelled by their dynamic and flexible development environment. Manual testing works best in these situations, where frequent changes and rapid iteration are key. Its flexibility allows testers to quickly identify errors and adjust to changing requirements. This is especially helpful for smaller organizations with limited resources, as they may not be able to make the initial investment necessary for an automation testing framework.

Mid-sized companies:

Finding a happy medium using a mix of strategies. Mid-sized companies can benefit from a hybrid approach that takes the right path between complex assignments and resource constraints and combines manual and automated testing where human sensitivity and flexibility are needed great, such as functional testing, analytical testing, and handling edge issues Manual testing is inevitable At the same time, working with monotonous features such as performance and regression testing increases performance great, and provides features that allow human testers to focus on complex issues.

Large projects:

Automation is used to increase productivity. Larger companies struggle with the limitations of manual testing because they manage large systems and tests. The development timeline can have a significant impact on the time required to conduct manual testing, and scalability is a major concern. In these cases, automation testing emerges as an effective solution. Larger projects can achieve faster response cycles and be more efficient by automating common tests. In addition, automated tests reduce the chances of human error by ensuring consistent and reliable execution.

Beyond dimensions: Different research approaches

Although project scope is an important factor in selecting appropriate test methods, several test methods should be considered to achieve adequate test coverage the study highlighted which test methods emphasize the following importance:

- Unit testing is a method used by developers to test the functionality of individual software units. Usually automated. This setting is necessary to ensure that the installations in the application work as intended.
- Integration testing is the process of confirming how components interact with each other. Depending on the complexity of the integration, manual and automated testing may be required.

- Functional testing ensures that application features meet specified requirements. Functional testing includes automated testing and manual testing, which supports various approaches such as white box and black box testing.
- Passive testing: This refers to methods for assessing passive characteristics, including usability, safety, performance, and availability Passive testing uses human testing and testing they use all their own.

Integrating strategies: Developing a common assessment methodology

The strategic integration of testing methods is necessary to achieve an integrated testing approach. The most important things to consider are:

- **Test Plan:** A comprehensive test plan detailing features, scope, and test procedures is required. This system must stay adaptable to meet the demands of a changing industry.
- **Equipment selection:** Choosing the right equipment for both manual and automated testing is important. While choosing the right framework for automation testing is important, defect detection and test management solutions accelerate manual testing.
- **Collaboration:** It is important that the craft and laboratory communicate well and work together. Discussion, identification of limitations, and exchange of information enhance the overall testing effort.
- **Continuous Improvement:** Test methods should be flexible, changing as industry and technology evolve. Strategies are constantly explored to ensure adaptability to changing circumstances.

In summary:

In conclusion, due to the dynamic nature of software development, software testing should take a sophisticated and flexible approach. The objective of this comparative study was to shed light on the advantages and disadvantages of manual and automated testing approaches and enable software development teams to adapt their approach according to project objectives and project scope.

Organizations navigating the complex world of quality can benefit greatly from the comprehensive guidance provided by the combination of test methods and design concerns presented in this study. Software development teams are able to understand the unique characteristics of each testing methodology and strategically combine them to create an environment for successful, efficient, and high-quality software delivery One important finding was the need for considerable work is required in identifying optimal testing protocols.

As software development progresses, testing will always be necessary to ensure the quality of the final product. Because applications, designs, and technologies are constantly changing, testing methodologies must be flexible and appropriately integrated. The concept of effective collaboration, continuous improvement, and a thorough understanding of the contextual factors influencing testing decisions is a way to explore sustainable integrated testing processes.

6.2 Recommendations for Future Research

Although this research thoroughly understands human and laboratory capabilities, there are still areas for future research that can deepen our understanding of software testing methodologies. Suggestions include:

- **Analysis of Industry-Specific Bottlenecks:** Analysis that analyses nuances and industry-specific bottlenecks in human and mechanical testing provides highly focused insights for companies involved in various industries.
- **Application of Machine Learning in Software Testing:** Research on machine learning techniques in software testing can lead to solutions designed to automate complex testing scenarios to improve the testing.
- **Analysing the impact of emerging technologies:** Considering how cutting-edge technologies such as blockchain, artificial intelligence and the Internet of Things impact testing methodologies can provide an idea of how the software development industry is changing around.
- **Analysis of Organizational and Cultural Factors:** A better understanding of the contextual and human aspects of testing can come from examining how

organizational and cultural factors influence the selection and effectiveness of testing strategies in the influence of.

- **Evaluate the long-term effects of testing strategies:** Analysis of the impact of a particular test system on software maintenance, scalability, and overall system resilience over time will shed light on how strategies a use is on the sustainable.

6.3 Closing Remarks

This comprehensive review examined the challenges of software testing, exploring the benefits and limitations of both manual and automated testing we have found that there is no one-size-fits-all solution, and the best testing methodology is commercial. Reflects a variety of factors including size, type of project and resource constraints.

Here are why the insights gained from this study are so valuable to practitioners, researchers, and organizations:

- **Accreditation:** Software development testers take the lead in ensuring software quality. By understanding the strengths and weaknesses of manual and automated testing, testers can choose the most appropriate methods for a particular testing environment which this empowers them to optimize test methods, achieve testing detailed information, and finally provide high-quality software.
- **Guidance for researchers:** Researchers in the software testing field can use the findings of this study to identify areas for further research. The growth of AI, machine learning, and DevOps techniques requires continuous research and testing practices. This study can serve as a catalyst for future research that examines how to incorporate this emerging technology into testing procedures.
- **Reporting organizations:** Software development organizations are constantly faced with the challenge of balancing quality with effectiveness. This study provides valuable insights that can help organizations choose the most appropriate testing strategy based on their specific needs. By understanding the advantages and disadvantages of both manual and automated testing, organizations can optimize their testing processes, improve product allocation,

and ultimately deliver consumer-friendly software meets role expectations and meets performance objectives for.

A changing landscape and the need for continuous improvement

The software development landscape is constantly changing, with advances in technology, evolving user requirements, and the adoption of new methodologies such as Agile and DevOps as these paradigms change, testing practices must change accordingly.

This study highlights the importance of continuous improvement in software testing. By studying emerging trends, such as AI-driven test automation, machine learning for fault prediction, and specialized testing methodologies for IoT applications, organizations can ensure that their testing processes remain robust and fly effective in the face of change.

Concluding Thoughts

In conclusion, this review provides a comprehensive review of manual and automated testing methods. By emphasizing the importance of testing strategies tailored to specific needs and embracing continuous improvement, this review empowers practitioners, researchers, and organizations to navigate an ever-evolving world of software testing of the and offer exceptional software experience they will. As the future of software development unfolds, the insights presented here will serve as a valuable roadmap for ensuring software quality remains a top priority.

7. References

AALS, van der, 2019. Automated GUI Testing. . 2019.

ADIL, Mohammad, 2024. Challenges In Automation Testing. . 2024.

ANIL KUMAR, ANUJ KUMAR, and Neeraj Kumar., 2017. Recent Trends in Software Testing. *International Journal of Advanced Research in Computer Science and Software Engineering,*. 2017. Vol. 7, no. 10, p. 301–306.

ANNE METTE JONASSEN HASS, Lesley M, 2016. Global Software Test Automation. . 2016.

ARBON, J., & BURNETT, M., 2012. Model-based Testing. . 2012.

BACH, J., & BOLTON, M., 2010. Exploratory testing. . 2010.

BELL, J., 2018. Better Software Test Automation. . 2018.

BHATIA, S., & SHARMA, A, 2018. Comparative Study of Manual and Automated Testing Tools. *Journal of Scientific Research in Computer Science*. 2018. Vol. 3, no. 3, p. 1- 8.

BINDER, Robert V., 2018. Testing Object-Oriented Systems. . 2018.

BINDER, Robert V, 2013. Automated Software Testing. . 2013.

BOT PLAY AUTOMATION, 2022. Automation & Manual Testing Best Practices - Improve Software Quality. . 2022.

CEM KANER, JACK FALK, Hung Q. Nguyen, 2008. Testing Computer Software. . 2008.

COPELAND, Lee, 2004. A Practitioner’s Guide to Software Test Design. . 2004.

DANIEL SUNDMARK, Paul Pettersson, 2019. A Study of Manual and Automated Testing for Industrial Control Software. . 2019.

DESIKAN, Srinivasan, 2006. Software Testing: Effective Methods. . 2006.

DILMEGANI, Cem, 2024. Automation Testing Case Studies. . 2024.

DRUSINSKY, Doron, 2017. Software Testing and Continuous Quality Improvement. . 2017.

ENOIU, Eduard Paul, 2017. A Comparative Study of Manual and Automated Testing for Industrial Control Software. . 2017.

GAIKWAD., Amol S. Dange and Suhas A., 2020. Recent Trends in Software Testing. *International Journal of Scientific & Engineering Research*. 2020. Vol. 11, no. 4, p. 178- 181.

GARG, Varsha Choudhary and Deepak, 2018. A Review on Recent Trends in Software Testing. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2018. Vol. 8, no. 4, p. 40–47.

GAROUSI, V., KHEZRIAN, M., FELDERER, M, 2018. A Review and Comparison of Automated Testing Tools for GUI-based Applications. *Journal of Software: Evolution and Process*. 2018. Vol. 30, no. 3.

GOYAL, A., & SINGHAL, M., 2018. An Overview of Ad-hoc Testing. *International Journal of Engineering & Technology (IJET)*. 2018. Vol. 7, no. 4, p. 119–124.

GRIGERA, PABLO, and Santiago Matalonga, 2007. Practical Regression and System Testing. . 2007. Vol. 24, no. 2, p. 101- 103.

HAMILTON, Thomas, 2024. Difference Between Manual and Automation Testing. . 2024.

HUANG, Y., HUANG, G., LEUNG, H, 2019. Generations of Automated GUI Testing. . 2019.

JACKSON-BARNES, Shannon, 2024. Testing Trends for 2024 and Beyond. . 2024.

JOHNSON, A., 2019. A Comprehensive Guide to White Box Testing Techniques. *Journal of Software Engineering*. 2019. Vol. 14, no. 3, p. 102–115.

JORGENSEN, Paul C, 2002. Software Testing. . 2002.

JORGENSEN, Paul C., 2013. Software testing: a craftsman's approach. . 2013.

K., Anupam, 2022. evaluate the performance of an automated testing framework. . 2022.

KANER, CEM, and James Bach, 2002. Automated versus manual testing. . 2002. Vol. 12, no. 2, p. 125–147.

KATAR´INA HRABOVSK´A, BRUNO ROSSI, and Tom´aˇsPitner, 2019. Software Testing Techniques. . 2019.

KHIN SHIN THANT, Hlaing Htake Khaung Tin, 2023. THE IMPACT OF MANUAL AND UTOMATIC TESTING ON SOFTWARE TESTING EFFICIENCY AND EFFECTIVENESS. . 2023.

KUMAR, S., & SHARMA, R, 2021. A Comparative Study of Various Manual Acceptance Testing Methods. *International Journal of Software Quality Assurance and Testing*. 2021. Vol. 9, no. 4, p. 32–45.

KUMAR., Shweta Chaudhary and Alok, 2021. A Survey on Recent Trends in Software Testing. *International Journal of Computer Sciences and Engineering*. 2021. Vol. 6, no. 8, p. 59–65.

KUMAR, ankaj Kumar and Umesh, 2019. A Review on Recent Trends in Software Testing. *International Journal of Engineering and Advanced Technology*. 2019. Vol. 9, no. 3, p. 187–191.

KUMARI, Bhawna, CHAUHAN, Naresh and VEDPAL, 2018. A COMPARISON BETWEEN MANUAL TESTING AND AUTOMATED TESTING. *Journal of Emerging Technologies and Innovative Research (JETIR)*. 2018. Vol. 5, no. 12, p. 2–9.

MUBARAK ALBARKA UMAR, Zhanfang Che, 2019. A Study of Automated Software Testing Automation Tools. . 2019.

NGUYEN, H., & TRAN, T., 2014. Case Study: Improving Test Efficiency through Automated Testing in a Mobile Application Development Environment. *International*

Journal of Computer Science and Information Technology. 2014. Vol. 6, no. 2, p. 89-97.

NGUYEN, T. T., & KAPFHAMMER, G. M., 2021. Automated GUI testing for mobile applications. . 2021. Vol. 172.

NGUYEN, T.T., MEMON, A.M, 2017. A Comparative Study of Manual and Automated Testing Techniques for GUI-based Applications. . 2017.

PAI, Akshay, 2023. Manual Testing vs Automation Testing. . 2023.

RAFI, Dudekula Mohammad, 2021. Benefits and_limitations of automated software testing Systematic literature review and practitioner. . 2021.

RAJESH KUMAR MISHRA, 2017. Future Trends in Software Testing. *International Journal of Computer Applications*. 2017. Vol. 168, no. 5, p. 0975 – 8887.

RASHID, F., MOHAMAD, R. A., & ALSARAYREH, S, 2019. A Comparative Study of Automated GUI Testing Tools. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*. 2019. P. 453–458.

REHKOPF, MAX, 2022. Automated software testing. . 2022.

SABA KHALID, Muhammad Usman, 2021. Emerging Trends and Challenges in Software Testing: *International Journal of Advanced Computer Science and Applications*. 2021. Vol. 12, no. 1, p. 246–253.

SAHA, D., ROY, C. K., & KIM, D. S., 2020. Automated GUI Testing Techniques and Tools. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2020. P. 682–686.

SHARMA, A., & GUPTA, R, 2020. Gray Box Testing Techniques. *Journal of Scientific Research in Computer Science*. 2020. Vol. 5, no. 1, p. 31–36.

SINGH, Anshuman, 2023. What is the Difference Between Manual and Automation Testing? . 2023.

TALHA AHMED KHAN, 2021. A Survey on Recent Trends in Software Testing.

International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2021. Vol. 6, no. 1, p. 137–144.

TED KURMAKU, Musa Kumrija., 2020. A SYSTEMATIC LITERATURE REVIEW AND METAANALYSIS COMPARING AUTOMATED TEST. . 2020.

THAKKAR, Mit, 2024. Deep Learn about different types of Manual Testing. . 2024.

TIAN, Jeff, 2005. Software Quality Engineering. . 2005.

TULLIS, T. S., & ALBERT, B., 2013. Measuring the User Experience. . 2013.

VAHID GAROUSI, Mika V, 2016. When and what to automate in software testing? . 2016.

WHITTAKER, James A, 2008. Software testing techniques. . 2008.

8. List of Tables, Figures and Abbreviations

8.1 List of Table

8.2 List of Figure

8.3 List of Abbreviations

QA:	Quality Assurance
UI:	User Interface
UX:	User Experience
KPI’s:	Key Performance Indicators
BAT:	Business Acceptance Testing
SEO:	Search Engine Optimization
MLR:	Multivocal Literature Review
SLR:	Systematic Literature Review
GUI:	Graphical User interface
C&R:	Capture and Replay
MBT:	Model based Testing
SUT:	System Under Test
ANOVA:	Analysis of Variance Formula
ROI:	Return On Investment
IT:	Information Technology

SDLC: Software Development Life Cycle

CRS: Customer Requirement Specification

SRS: Software Requirement Specification

DDS: Design Document Specification

STLC: Software Testing Life Cycle Technique

UAT: User Acceptance Testing

UFT: Unified Functional Testing

QTP: Quick Test Professional

OWASP ZAP: Open-Source Web Application Security Scanner

REST API: Representational State Transfer Application Programming Interface

SOAP: Simple Object Access Protocol

XML: Extensible Markup Language

URL: Uniform Resource Locator

POM: Page Object Model

API: Application Programming Interface

TestNG: Test Next Generation

CI/CD: Continuous Integration / Continuous Deployment

ML: Machine Learning

MNCs: Multinational Corporation

DevOps: Development and Operations

AI: Artificial Intelligence

IoT: Internet of Things

Appendix

Questionnaire

Demographics

1. Age

- Below 25 years
- 26 to 30 years
- 31 to 35 years
- 36 to 40 years
- Above 40 years

2. Gender

- Male
- Female

3. **Years of Experience**

- <1 year
- 1-3 years
- 3-5 years
- 5-10 years

4. Job Role

- Test Analyst
- Manual Test Engineer
- Automation Test Engineer
- QA Manager
- Software Developer
- Test Project Manager

5. Industry

- Finance
- Healthcare
- Technology
- Manufacturing
- Education

6. Company Size

- Small (<50 employees)
- Medium (50-250 employees)
- Large (>250 employees)

This section consists of a structured questionnaire based on a 5-point uniform scale. In this, 1 strongly disagree, 2 disagree, 3 neutral, 4 agree, 5 strongly agree.

Defect Detection Rate:

Statement	1	2	3	4	5
Manual testing is more effective than automation in detecting defects.					
Automation tools can detect defects more efficiently than manual testing.					
The defect detection rate is higher in automated testing compared to manual testing.					
Manual testing allows for better identification of subtle defects compared to automated testing.					
Automated testing helps in detecting defects across different platforms and configurations better than manual testing.					

Test Coverage:

Statement	1	2	3	4	5
Manual testing provides better test coverage compared to automation.					
Automation tools ensure broader test coverage than manual testing.					
The test coverage achieved through automation is more comprehensive than manual testing.					
Manual testing allows for more precise targeting of critical areas for testing compared to automation.					
Automation enhances test coverage by executing repetitive tests across various scenarios.					

Testing Efficiency:

Statement	1	2	3	4	5
Manual testing is more time-consuming and less efficient compared to automation.					
Automation tools significantly improve the efficiency of testing processes.					
Manual testing requires more effort and resources compared to automation.					
Automated testing reduces the time required for regression testing compared to manual methods.					
Automation enables quicker feedback on changes, enhancing overall testing efficiency.					

Cost Effectiveness:

Statement	1	2	3	4	5
Manual testing is more cost-effective than automation.					
Automation tools require substantial initial investment but prove cost-effective in the long term.					
Manual testing incurs higher costs due to increased resource requirements.					
Automation reduces overall testing costs by minimizing human involvement.					
The initial cost of setting up automation may be higher, but it leads to cost savings over time.					

Tester Satisfaction:

Statement	1	2	3	4	5
Testers prefer manual testing over automation.					
Automation tools contribute to higher job satisfaction among testers.					
Manual testers feel more engaged and in control of the testing process.					
Automation reduces mundane tasks, leading to increased satisfaction among testers.					
Testers find automation tools user-friendly and enjoyable to work with.					

Quality of Testing Documentation:

Statement	1	2	3	4	5
Manual testing results in more thorough and detailed documentation compared to automation.					
Automated testing generates more consistent and standardized testing documentation.					
Documentation quality is compromised when using automation tools.					
Manual testers tend to provide richer contextual information in test documentation.					
Automation tools facilitate easier maintenance and updating of testing documentation.					