Efficient Software Testing for Assuring Software Quality

Dawan Minhajul Islam (17-33608-1)

Tabassum Nazia (17-34479-2)

Abu Hena Mostofa Kamal (17-34942-2)

Antu Asif Ikbal (17-34554-2)

A Thesis submitted for the degree of Bachelor of Science (BSc) in Computer Science and Engineering (CSE) at

American International University-Bangladesh
in June, 2022

Declaration

This thesis is our unique work, and it has not been submitted in any manner for another graduate certificate at any university or other educational institution. The book acknowledges and provides a list of references for information drawn from others published and unpublished work.

Dawan Minhajul Islam
17-33608-1
CSE

Tabassum Nazia
17-34479-2
CSE

Abu Hena Mostofa Kamal
17-34942-2
CSE

Antu Asif Ikbal
17-34554-2
CSE

Abstract

Software testing is essential before releasing software since everything must be verified for problems. Software testing, on the other hand, is an expensive and time-consuming process. This post will look at both manual and test automation. We demonstrate that automated testing is quicker than testing process and that testing phase is shorter than automation testing in this article. During the testing process, testing phase also saves time and money. Software testing is the process of evaluating a software package and looking for faults or defects in order to generate defect-free software. The only way to judge a program's quality is to put it to the test (software testing). As technology has improved across the world, the number of verifying techniques and ways to test software before it goes into production and, of course, to market has grown. Automation testing has had an influence on the testing process. The bulk of software testing is now performed using automated tools, reducing both the number of humans working around the software and the number of defects that the tester's eyes may overlook. Test cases make it easy to collect and handle a variety of scenarios in automated testing. As a result, software automation testing is important to the success of software testing. The purpose of this study is to learn about different types of software testing, as well as testing methodologies and tools, as well as to compare manual and automated testing. Test automation is essential in today's fast-paced agile development environments. By reducing test execution cycles, the main goal is to reduce the amount of work necessary to manually conduct tests. By applying test generation to a GUI-based application built for a significant industrial project, we pushed test scripts to the next level. This book examines the transition from manual to automated procedures. We've got you covered on everything from test cases to automating GUI test design. The following are crucial lessons to remember: Automated testing ladder, which was just proposed In agile development, automated test creation does not lessen the need for high-level GUI testing, and automated test generation does not lower the need for high-level GUI testing. In practice, the time and effort necessary to review the results of generated tasks limitations the usage of unit testing creation. It covers a wide range of topics, including how to successfully apply leadership development programs in a genuine industrial project.

Approval

The thesis titled "Efficient Software Testing for Assuring Software Quality" has been submitted to the following respected members of the board of examiners of the department of computer science in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering on (8-06-2022) and has been accepted as satisfactory.

Md. Mazid-Ul-Haque
Lecturer & Supervisor
Department of Computer Science, American International University-Bangladesh

Akinul Islam Jony
Assistant Professor
Department of Computer Science, American International University-Bangladesh

Dr. Md. Abdullah - Al – Jubair
Assistant Professor and Head-in-charge (UG)
Dept. of Computer Science FST, American International University-Bangladesh

Professor Dr. Tafazzal Hossain
Dean, FST
American International University-Bangladesh

Dr. Carmen Z. Lamagna
Vice Chancellor
American International University-Bangladesh

Acknowledgement

First and foremost, we express our gratitude to Almighty Allah; without His wonderful gift, we would not be able to finish our thesis. We would like to thank our supervisor, **Md.Mazid-Ul-Haque,** Lecturer, Department of Computer Science, American International University-Bangladesh, for his patience, guidance, and support during the process of this research. He's been encouraging since we started working on the thesis. He has always been willing to assist and offer his best recommendations, and we consider ourselves fortunate to have his direction and support throughout this thesis. We owe a debt of gratitude to all of our respected professors, without whom we would not have progressed to this level in our learning. We appreciate their time and thoughts, as well as their willingness to help us anytime we needed it. We'd want to thank our parents for their patience.

Table of Content

Table of Figures

List of Tables

.

Chapter 1: Introduction

1.1 Introduction

Everything in the current era is dependent on ever-expanding technology. Everything in today's world is dependent on software, which is why it needs to improve. To make business easier, effective software is required. To make software more efficient, two types of testing are required: manual testing and automated testing. There are various distinctions between human and automated testing. Manual test is one that is completed step by step without the use of a test. The use of a range of tools and software to execute tests that are run automatically through test automation is referred to as automated testing. Both human and automated testing have advantages and disadvantages. Hand testing takes longer than computer testing. Both have advantages and disadvantages. Manual testing is time-consuming and inconvenient. On the other hand, its strength is that it is better at dealing with complex difficulties. Upkeep of code and tests is required for automated testing. On the plus side, it's a lot quicker and covers a lot more ground [1].

The purpose of the software quality approach is to produce methodologies that allow the software industry to offer safe, dependable, and usable software solutions while maintaining within a budget that allows enterprises to compete effectively. Meeting this challenge will have a major impact on future economies, global security, greater competitiveness, and overall well-being. Software quality considerations include accuracy, robustness, scalability, and evolvability, to name a few. There is no globally accepted definition of product testing, and no agreement on how to adequately quantify any of the major quality issues. The actual grade of success achieved in practice is determined by quality evaluation products and tools, including the willingness to pay for their utilization. The process of developing software in accordance to a set of regulations is known as software development. To check and validate that the software generated fits these standards, software testing is necessary. Software testing assists in the avoidance of system failures. It refers to the process of evaluating software in order to find faults. Usability, compatibility, stability, integrity, efficiency, protection, capability, portability, and maintainability are also checked in software. Software testing aims to meet specific goals and principles that must be followed. In its most basic form, testing is a method of finding flaws in software. Software testing comprises executing software to verify functionality,[2]

i. Verification: This is the procedure for checking that the software complies with the specifications.

ii. Detection Approach: This is the practice of doing wrong inputs on purpose in order to evaluate the system's operation.

iii. Validation: Validation is a process of verifying that software satisfies the requirements of the client. The author is from the Department of Computer Science and Engineering at Bangladesh University of Business and Technology.

Testing is a technique for establishing whether or not the completed system meets the requirements specified at the start. It's usually a system that has to go through a validation and verification process to see if the developed system meets the customer's requirements. As a result, in this activity, the difference between the actual and projected outcome is drawn. As a consequence, this is an analysis that provides associates with the essential product quality information. Software testing is frequently regarded as a high-risk undertaking. Figure 1 depicts the relationship between testing expenses and errors. It appears to have been established that the cost of testing rises dramatically (functional and nonfunctional). The compelling testing goal is to do as few tests as feasible in order to prevent unnecessary testing. From Figure 1, we can deduce that software testing is an important part of software quality assurance. Software testing's main purpose is to create a high-quality product in terms of estimation of dependability, comprehensive verification, and validation. Testing's secondary purpose is to execute a program in order to find flaws and construct a test case capable of detecting the error that has yet to be found. Software must be adequately tested to ensure that it will perform as expected in its intended environment. Software testing must be successful in terms of detecting errors, but it must also be efficient in terms of completing tests as fast and efficiently as feasible. Software testing automation can considerably minimize the work necessary for appropriate testing or greatly increase the quantity of testing that can be completed in a given amount of time. Tests that would normally take hours to complete can now be completed in minutes. manually. The literature review in this book highlights how various organizations have always been able to program testing, with some saving large amounts of time and money [3], [4].

large sums of money It has proven possible to save up to 80% of the time spent on manual testing. Some businesses have not immediately saved money or effort. However, test automation has enabled them to generate higher-quality software more quickly than human testing would have allowed. Testing at the 'press of a button,' with tests executed overnight when machines would be idle, is possible with a well-developed test automation regime. Test suites are repeatable because they are using the same inputs in the same order every time, which is impossible to ensure with manual testing. testing by hand with automated testing, even the tiniest maintenance changes may be thoroughly evaluated with minimum effort [4].

1.2 Motivation

The goal of software testing may be defined in a variety of ways, but the common thread is that it ensures that software functions as expected and as intended. Finding faults or flaws that may persist in the program throughout development, preventing defects, and ensuring that the finished product fulfills business and requirement specification are all responsibilities. Technology's impact on our career and personal life keeps evolving at a breakneck speed. The digitalization is expanding to embrace every facet of human existence, from mobile apps that control appliances to virtual reality. Businesses are launching smartphones that are used by hundreds, if not millions, more people all around the world. The bulk of those companies are employing the Agile quick delivery framework to create software, which translates to about one new release every two weeks. Before each launch, such apps must be thoroughly tested to ensure that the end user has the best possible experience. Manual testing is absolutely insufficient at such speed. Every area of the economy like Banking, agriculture, transportation and healthcare, nearly now depend on software. That's why making better software needs testing for testing can use automation or manual testing.

The software industry, which was thriving in the latter decade, has suffered a huge setback in the new century. So, when Year 2000 problems faded, the IT industry saw a significant slowdown. Hundreds of new businesses vanished as a result of closures and mergers, and the industry began to consolidate. Major reorganization began in multinational corporations all over the world. The last three years have been a period of stabilization, and one positive outcome has been the awareness of the value of software quality. Quality assurance objectives, which are comparable to test plans, are developed by many software quality organizations. A software testing plan, on the other hand, may incorporate activities not covered in a testing phase. The test plan is an element of the continuous improvement cycle and is one of the principles of quality management, despite the fact also that the quality assurance plan contains the full quality spectrum. It consumes a lot of effort, dedication, and money to build a skilled devoted QA crew. We must strike a delicate balance between how much money we spend on bringing in resources and how much money we spend on a smart tool that can help with testing [3]–[5].

process by certain experts. So, we want to introduce a new, easier hybrid model that combines the advantages and disadvantages of Waterfall and XP model.

The most well-known and oldest SDLC model is the Waterfall Model. This model is frequently used by the government as well as a number of significant enterprises. The sequential steps are a unique aspect of this model. It's also the first plan-driven approach to prioritize early planning, requirement analysis, and process design. It also improves software quality and stability by reducing the need to adjust to new requirements after deployment. As a result, the main concerns in this model are guaranteeing a flaw-free design and requirement analysis. The Waterfall Model is a controllable strategy that is appropriate for lifecycle management of smaller projects with needs that are identified and finalized up front. On the other hand, this model requires the completion of the previous step before moving on to the next. The Waterfall Model should only be used with projects that do not predict unforeseen changes in the middle of development because it does not allow returning to previously completed phases. So, the Waterfall Model does not perform well for complicated projects with the possibility of changing requirements and/or extensive impromptu testing during the software development stage because of its strict structure.

One of the Agile techniques is Extreme Programming. It adheres to all Agile principles, including significant client involvement in the software development process, effective team communication, and iterative development cycles. Many ground-breaking software development regulations have been developed as a result of XP. It can deal with huge development budgets and build effective innovation processes to improve software performance. Extreme Programming is a practice that allows software development organizations to save money and time when completing projects. Because XP is focused on the timely delivery of finished products, time savings are available. Because they don't utilize a lot of documentation, Extreme Programming teams save a lot of money. They frequently handle difficulties by debating them within the team. Another benefit of Extreme Programming projects is their simplicity. Developers who favor this approach produce extremely simple code that may be improved at any time. Because of the regular testing during the development stage, XP helps to generate software faster. On the other hand, Extreme Programming is more concerned with the code than with the design. This could be a concern because software applications require excellent design. It makes it easier to sell them on the software market. Furthermore, defect documentation in XP projects isn't always up to par. Due to a lack of defect documentation, similar defects may emerge in the future.

In current times, Agile technology is very popular in Bangladeshi software companies. XP is one of the popular Agile models for Bangladeshi software companies. Companies nowadays use XP for some good features. On the other hand, the Waterfall model is also very popular but its popularity among Bangladeshi companies is decreasing day by day. Waterfall model also carries some good features. So, we want to introduce a hybrid model that combines the best features of

Waterfall and XP models. We want to show that this hybrid model will be easier to use compared to the other two individual models.

1.3 Problem Statement

The overall goal of software test is to validate that the end user receives a high-quality product. It is necessary for high-quality software to have a low probability of mistakes and faults in production. Software development is complicated and prone to faults at any level. If these problems or flaws are not found and fixed as soon as feasible, they can result in significant time and financial losses. Software testing guarantees that software fulfills user specifications and needs, and that it performs as intended. In any event, there are a number of challenges in the field of testing process that must be addressed, such as the efficient development of testing data or detailed test, the prioritizing of test data, and so on. Testing is an important aspect of the software life cycle. Testing in an effective and timely way may assist in resolving issues quickly and ensuring software stability. When a program's executing deviates from its intended behavior, it is called a software failure. As a result, failure (and, by extension, error) is a relative concept [6]. Errors and failures occur when the original plan is not followed. Without a clear grasp of what was meant, it is impossible to be positive that a program's performance has strayed from its planned intent, and therefore that a failure or error has occurred.

1.4 Objective

The objectives of our research work are

- To analyze the current situation of the Efficient software testing for assuring software quality.

- To identify the automation and manual testing performance and evaluating the methods and technologies for integrating latent quality into software over post-development testing.

- To find out the most efficient system's quality assurance, as well as how automation testing reduces test costs and boosts job productivity.

1.5 Research Outcome

This study's findings show that automated testing must have at least two major advantages over test cases: efficiency and accuracy. Determine the value of using automated testing software to regulate and improve quality. Define the Benefits of Automated Testing Compared to Manual Testing.

1.6 Research Outline

Background research is conducted to determine the primary purpose of the study; research methodology is conducted to determine how the study will be discussed; discussion analysis is conducted to determine the comparison of testing; and based on the results of the analysis, we have accomplished our research objective.

Chapter 2: Background study

2.1 Literature review:

The goal of the quality assurance study is to develop tools and technologies that will enable the software industry to produce really safe, durable, and usable software and services while staying cost-effective and allowing businesses to compete effectively. The impact of addressing this issue on national economies, national security, global competitiveness, and general well-being will be enormous. Correctness, robustness, readability, and evolvability are only a few examples of software quality considerations. There is no widely accepted metric for software quality, and there is no consensus on how to effectively quantify any of the primary difficulties. Quality evaluation procedures and technologies, as well as the readiness to pay for their use, influence the true quality standards achieved in practice [6]–[8].

Working with high-quality software is something that everyone wishes for. Managers understand the importance of good quality, software developers understand the benefit of producing high-quality software, and consumers want software to perform consistently and reliably. The Quality Assurance Institute (QAI) and the American Software Quality (ASQ) Institute outline a number of quality assurance approaches that are used by a range of enterprises[6]–[8]. To enhance and investigate their software systems, some companies create software quality assurance groups. Quality assurance, on the other hand, is not a widely acknowledged process. As a result, quality assurance groups in different companies may have distinct tasks and use different ways to achieve their goals. Software testing is a collaborative activity in certain companies. Testing software is sometimes the job of a developer team or a separate company.

According to several studies, quality assurance activities are prioritized 50–60% of such time while designing large software systems. The fraction might be significantly greater in life-threatening conditions, such as testing. Software-based systems It's critical to get the most rapid and effective leverage possible. Simply put, by bridging the gap between previous research and current practice. Practitioners are only aware of a limited fraction of the research community's work, and only a low percentage with that work ever has been published. Software quality assurance plans, which are comparable to test plans, are created by many software quality assurance businesses. A software maintenance plan, on either hand, may contain information that a test strategy does not. The test plan is a component of the quality assurance plan one of the product testing instruments, despite the fact that it covers the complete quality spectrum [9]. A software failure occurs when a software program fails to perform as planned. just as expected A software fault is a problem in software that causes it to fail, whereas an error is a flaw in human performance. the capacity to comprehend and carry out in a way that results in the process of introducing a defect into something A software breakdown is derived in the execution of an action when carefully studied. from the planned behavior of the software as a result, the notion of failure (and, by extension, the notion of error) is subjective. Errors and failures occur when the original plan is not followed. It's tough to make judgments without a clear image [9]. As little more than a

result, the under-review program is weakened. Less formal methods, such as the bulk of testing and analysis, can, nonetheless, be helpful. It may be used to identify dependencies, handle with non-functional actions, and set targets, for example [2]. Controllable restrictions on quality assurance jobs Human involvement in the process These fewer formal strategies can be used to address a wide range of issues. They range in complexity from simple to complicated, and the methods upon which they are built are varied. Under the worst scenario, complexity varies from linear to hyper exponential [1].

Manual testing is a method of detecting software defects in which the testing manually writes and performs test cases. It is the most difficult and time-consuming method of testing. Testing phase is a time-consuming technique that requires patience, attention, speculation, creativity, imagination, open-mindedness, and expertise from the tester. Repetitive manual testing on large software programs or apps with a large dataset coverage might be difficult. The initial phase of testing focused on writing test scripts and manual allows us to take a look at the graphical user interface [9]. JUnit tests are used to create unit tests. are used for regression testing in non-UI programming. Daily builds and continuous integration tests They had to be the ones. This is the foundation for our test automation. The bulk of personality characteristics, on the other hand, The computer keyboard that was described in the application criterion had to be put to the test. programming interface (GUI) level (GUI) For a sample application, a prototype Interface was created. for the parts of a standard keyboard, it was decided to utilize the demonstration app. to aid the team with manual testing process and to demonstrate the progress of growth by indicating which features have been introduced and providing guidance on how to utilize them Writing test scripts in scripting languages like Python, JavaScript, or Tcl (Tool Command Language) to automate automated testing entails writing test scripts in scripting languages like Python, JavaScript, or Tcl (Tool Command Language) so that possible solutions can be operate by computers with minimal human interaction. The test project plan might be managed in tandem can save time and money. Additionally, the automation program may feed test results into the unconfined aquifer system, compare predicted and actual outcomes, and provide complete test reports. Test automation involves a large investment of both money and effort. In succeeding development cycles, the same test script will also have to be run. Using an automated test tool, this testing technique may be captured and replayed as required. After the testing technique has been automated, no human interaction is required. The purpose of automated is to reduce the number of tests that must be completed manually, not to eliminate manual testing entirely.

Testing methods are used in automated testing to test software. (Total Operation / Total Effort (hr.)) Scripting for Automation The test team creates the script code using automation method to test software in this sort of testing; this statistic measures the automated testing script programming productivity [10]. This measure is used to assess how far automated testing has progressed. Software in automated testing, and test coverage must be calculated. Test coverage refers to how much of a manual process we can automate using testing tools. This measure is used to assess that however much automation a given test tool truly achieves. The percentage of hand test scenarios that

are automated is represented by this measure. (Total Number of Automated Test Cases/ Total Number of Manual Test Cases) *100For instance, when there are 100 conventional scenarios and 80 automated cases, Automated Covering equals 80%. The testing process takes a long time, is physically demanding, and requires a considerable financial expense in human resources. We can use automation tools to record the test suite and replay something if necessary. There is no need for social interaction after the testing procedure has been automated. Automation testing is more expensive up front than testing tools, and it is impossible to automate everything. Determine which test scenarios (human or automated) will deliver the highest return. Both automated and human software testing require metrics to assess their quality and success. The test measures give you a clear image of the product's quality and completeness, as well as insight into its readiness. The features of the system and services being developed are critical in deciding whether or not a satisfactory investment is attainable. Some companies assume that their projects or items are too small to be automated properly. It may and therefore should be disputed whether it is possible to obtain a satisfactory ROI before the evolution of tiny, short-term projects or items comes to an end. Setting up automation may be excessively expensive when comparison to the majority of something like the development process. Propose that the ratio of necessary automation resources to total resource availability be used to impact automation decisions. When working on new products or platforms, the corporation may not perceive adequate long-term opportunities [3]. This study, however, has yet to receive popular acceptance. Testing efficacy, according to 1970s experts, is defined as the percentage of statements and ties in an agency's flowgraph that were handled by several test executions. Such measures were considered as primitive and mass-produced in the 1970s. There are better measurements out there, according to a steady flow of studies. On the other hand, these study metrics are rarely discussed, and even fewer are employed in practice. System analysis does not appear to also be considered in practice. The software quality research community believes it has much to offer practitioners. Only a small fraction of theoretical concepts and techniques have ever found widespread application in practice, and the many of these ideas are at minimum 20 years old. For example, the testing phase was first proposed by researchers in the 1970s and is now widely used in practice. This type of testing is excessively expensive due to the human labor required to manage such testing elements as system testing, test controllers, and test results. Despite the fact that research has proven that much of this can be automated, and industrial apps are now available to help with these efforts, these approaches are rarely used.

Recent compilers seem to do an increasing number of studies, albeit they are often limited to object code optimization and automated array bounds checks. The outcomes of these assessments are rarely, if ever, made available to the user[10]. The Lint C analyzer is an outlier in the world of standalone static analyzer. There don't appear to be any standalone dataflow analyzers, and their purpose and utility are rarely recognized. In reality, there is indeed a lot of discussion about how to improve the software development process. The majority of process evaluation and assessment activities, on the other hand, do not place a premium on software product quality.

Despite the fact that process-based testing and analytical integration have great potential, the small traditional software technology department appears uninterested for doing so. Monitoring is an ability that may be acquired over time. Though it may come as a shock to some, it is an unavoidable truth. Even though any system can contain an infinite number of tests, we only have time to perform a small proportion of them in practice. However, determining which test cases to write and run is critical because this limited number of test scenarios is expected to uncover the majority of software flaws[10]. As we've discovered from both trial and experience, picking unit tests at random isn't a good way to test. If good test scenarios are to be developed, a more careful approach is required.

The solid lines indicate a testing process that I performed myself. When a test is automated for the first time, it becomes less adaptable and cost-effective. After a few runs, the device that automatically will be significantly less expensive than performing the same check manually. To build a strong and efficient suite of test automation, start with the most crucial component of a great test suite: a group of tests written appropriately by a tester that exercise the most significant components. The next step is to use automation to execute the tests, which will allow for reduced establishment and improvement. The solid lines show a testing process which I performed myself. When a test is automated for the first time, it will be less evolvable and cost-effective. After a few runs, the automated test will be significantly less expensive than doing the same check manually. To build a strong and efficient suite of automated tests, start with the most crucial component of a great test suite: a set of tests written appropriately by a tester to stress the most significant components. The next step is to use automation to automate the tests, which will allow for low-cost creation and implementation. To determine if the software getting tested operated properly, the results of each experiment must be examined [13]. It could be a short confirmation of the tester's expectations, or a lengthy and comprehensive comparing of the exact real outcomes with the predicted findings. While a test is running, you can examine some of the results, such as notifications broadcast on a screen. Other results, such as changes to database records, can only be evaluated that after test case is completed. A combination of these two strategies may be necessary in an automated test. If the actual and expected results are the same, the program has passed the test; when they're not, the software has passed. Then the software has failed the test. This is actually an oversimplification. All we can say is that if the actual and expected outcomes do not match, then something needs to be investigated. It may be that the software is incorrect, or it could be that the test was run in the wrong sequence, the expected outcomes were incorrect, the test environment was not set up correctly, or the test was incorrectly specified. There is a difference between comparing and verifying; a tool can compare but cannot verify [13].

A tool can compare one set of test outcomes to another, and can flag any differences between the two. But the tool cannot say whether or not the outcomes are correct this is verification. Might it be possible to automate the process of creating test cases? Testing tools can automate portions of test case design in a variety of ways. While these technologies, also known as test input generating tools, can be useful in some situations, they will never be able to totally replace intellectual testing. The tool may generate a

high number of tests, which is a problem with any test case design process. Some apps allow the tester to set a limit on the number of tests that can be created based on their requirements. The program, on the other hand, may generate much too many tests to finish in a reasonable amount of time. Because the algorithm is unable to determine which examinations are the most significant, creative intelligence is required. Considering that is nothing more than software, a tool can only obey commands [14].

A tool as well as a tester can both follow directions to execute a test case, but a human tester will approach the task differently if given the same task. If a human tester is in charge of performing a programmed test, for example, he or she can begin by following the steps and then double-checking the results. The tester, on the other hand, may discover that, while the program produces the desired results, both are incorrect. Furthermore, while the tests are running, the human tester can utilize his or her creativity and ingenuity to improve them. If challenges arise that are not part of the test case's continuous line, human testers outpace testing technologies again[11], [12].

A human tester, for example, will deal with the issue and do whatever is necessary if a network connection is lost and must be actually in the middle of a test. Occasionally, the tester will do this without knowing recognizing the test has deviated from its original path. An unanticipated event, on the other hand, can stop an automated test dead in its tracks. Of course, the technology can be programmed to respond to specific circumstances, but as explorers have seen, nothing trumps human ingenuity when it comes to problem solving. Automated testing program, although it does have a place as a source of data. started, as well as for specialized tasks. If you've invited folks in for a party, for example, and they manage to crash the system during 'playtime,' you can at least see what was typed in. In addition to accurate timing and performance statistics, the audit trail may provide a plethora of information not available from manual tests. Because putting up test data is a time-consuming and labor-intensive process, it is usually neglected, resulting in fewer test cases being executed than are required. Whereas if set of data can be captured the first time it is done, the tool can replay it much more reliably and usually lot faster if the user interface of the software does not change. It's not difficult to add the details of one new client, but entering the details of 100 new clients is best done automatically. Even if the next version of software makes user interface modifications that invalidate the recording, it may still be desirable to do so because the time saved during the first or second replay will quickly pay for the extra effort required to capture the data setup the first time. It is only true if there is no need to change the recording. When editing is required, the costs rise dramatically[13]–[15]. Another application for capturing manual testing, or at least human contact, is to apply the same maintenance update to a large number of data files or databases (s).

A recording can also be used to demonstrate something. A script for a multi-platform application could be recorded on one platform and then played back on a variety of others with little or no changes. Replaying complex sequences as a single step in a manual test can save time if they must be entered precisely and manually entering them is time-consuming. The test input is consistent because automated execution is significantly more precise than manual execution. When it comes to replicating bugs, exact input duplication is crucial. If the bug's cause has been noted, replaying it should

recreate the bug, and replaying it after it has been fixed should indicate if it has been fixed properly (or not).In most circumstances, however, automating test execution is futile. Manual testing is inefficient, time-consuming, and wasteful. When it comes to automated testing, however, some people end up with an actual test tool that generates piles of paper that must be constantly checked - this is lunacy. Test scripts are an important part of test automation for both interactive and non-interactive systems. The information in a script can be substantial, as we saw in the previous part, and the more info there is, the more things can go wrong, update, and manage[16], [17].

A screenplay is written in the same way that a computer program is written. Automating tests is an example of programming tests because most test execution platforms provide a scripting language that is effectively a programming language. Of course, creating tests is no less tough than creating perfect software programs, and it usually yields similar results. This means that no matter how good we are at programming or testing, the automated tests will have flaws. The more programming you do, the more bugs you'll find. It is vital to spend effort testing, troubleshooting, and repairing the tests in order for them to function successfully. This can take a long time, especially since it isn't restricted to first time the tests were automated. Changing the program frequently necessitates changing some of the tests, such as to accommodate new user interface options, menu modifications, user interactions, and deviations from expected outcomes. If it weren't for the requirement to write test scripts, test automation would be lot easier and faster. The more tests you automate, the more time and work advanced scripting techniques are worth investing [16]. The cost of producing and maintaining scripts, and the benefits we can obtain from using them, should be our primary focus, irrespective of how they are generated. If a script is going to be used by a variety of different tests over the course of a long period of time, it's critical that it performs well and is straightforward to maintain. If, on the other hand, the script will only be utilized for one test case and then abandoned once the test is over, it isn't worth the time and effort. The test instrument as well as the scripting techniques utilized will determine the content of a script. Scripts are incredibly adaptable, just like software. When writing a script to perform a given task, there are always several options. The way a script is written is usually dictated by the person writing its coding skills, but it should also be guided by the script's aim. Recording as much as possible or patching together bits of several scripts could be a 'quick and dirty' strategy. A more thoughtful approach can include both design and coding from the ground up. Some scripting techniques necessitate intricate logic and constructs, while others are far less difficult. One method is to write short scripts that each do a single action or activity that is repeated over multiple test cases. The same script can then be used in all test cases that need one of the common actions. Another technique to decrease scripting is to include control structures in the scripts that allow the tool to repeat sequences of instructions without having to code numerous copies. Although these methods will definitely result in the creation of more scripts at first, they will be lot smaller and hence easier to manage. However, after a sufficiently extensive set of scripts has been coded, this will happen eventually. Because scripts are such an important element of most test automation workflows, it seems reasonable to aim for excellence in them. Nobody wants to be connected with awful

scripts, can we be certain? The simple answer is "a script that is suited for purpose," however this isn't a very thorough response. Is this to say that a good script is one that reliably accomplishes its goals while being simple to use and maintain? Yes, but there are times when scripts do not need to be as simple to use or maintain, despite the fact that it is difficult to think of any. When scripts don't have to do what they're supposed to or are untrustworthy [16]. Much of what we have to say in this part will be of little relevance to you if you're not concerned with simplicity of use or maintainability for any of your scripts. We may identify a few qualities or attributes that will have a substantial bearing on how easy a script is to use and maintain, given that good scripts should be easy to use and maintain.

We'll evaluate two sets of scripts that perform the same set of test cases, rather than individual scripts. Occasionally, a tool seller will claim that their product generates 'simple to read' or even' self-documenting' recorded scripts.

Such assertions are both deceptive and shortsighted. Let us finally address these allegations. A 'raw' recorded script is made up of the entire sequence of test inputs, as well as some verification. Some tools, when used as directed by the user, can insert verification instructions while the recording is being done. The end product is frequently a large script with no discernible organization, sometimes hundreds of lines spread across multiple pages. It's similar to a book that doesn't have any chapters or section headings, just pages and pages of content. We normally need to look at a recorded script because we need to make a change to it, such as adding verification or control instructions, or changing a specific sequence of operations to reflect changes in the software under test. We'll need to discover the relevant area of the script in order to make these modifications correctly, and the longer the script is, the more difficult this will be. It would also be beneficial for us to obtain a comprehensive understanding of the script as a whole, so that any modifications we make do not have unintended consequences for other parts of the script or any other scripts that utilize it[18].If we are familiar with the scripting language, discovering things in a script and understanding it will be easier. It will also be simpler if the script includes some good documentation that explains what it does and how it does it. The most useful tool will be script annotation, which are comments embedded throughout the script that explain what is happening at each level. Some programs automatically include comments into the scripts they generate while recording, and this is sometimes what tool manufacturers mean when they say their scripts are "simple to read. “If we are familiar with the scripting language, discovering things in a script and understanding it will be easier. It will also be simpler if the script includes some good documentation that explains what it does and how it does it[19]. The most useful tool will be script annotation, which are comments embedded throughout the script that explain what is happening at each level. Some programs automatically include comments into the scripts they generate while recording, and this is sometimes what tool manufacturers mean when they say their scripts are "simple to read. “Words like 'Type,' 'Button,' and 'Click' are frequently used in scripts. Okay, but are we really looking for readability? We don't want 'difficult to understand' scripts that use acronyms and symbols instead of actual words, albeit this is debatable, as an experienced author may prefer common abbreviations and acronyms

for typing and reading convenience. The phrase "self-documenting" is a bit of a fiction. A script is documented when textual information is provided to assist or detail it. This is interpreted as more written information to what currently exists. The following are some examples of information that can be usefully written down:

user information what information must be passed to the script, what information it returns, what state the software under test must be in in when the script is called, and what state the software will be left in when the script ends).the raw content of the script the purpose of the script (what this script does, the actions it performs. A tool can only generate the first of the items above[18]. Only 'click the OK button,' 'focus on window "Payments,' and 'enter the string "John Doe" in the field "name" can be generated 'automatically.'

Because such information is frequently self-evident, annotations are rarely useful. If the script is to be used and maintained successfully and efficiently, this type of annotation is not enough. None of the other data can be generated without human intervention. If you want readable scripts, however, this information must be produced by someone, which is usually the script's creator [18].

This information should be included to the script as follows This individual will need to be able to edit the script itself, thus technical abilities are required (unless the program allows for easy comment input during script construction). If the people who are recording the scripts aren't tech-savvy, they'll need extra help.

Recording a lengthy test is not the end of the story; there is still much more work to be done. A recording, on the other hand, is frequently the starting point for test automation and, in certain cases, is sufficient. Keywords like 'Type,' 'Button,' and 'Click' are frequently used in scripts. Fair enough, however we don't want 'difficult to read' scripts that may use acronyms and symbols instead of correct words, albeit this would depend on the reader, since an experienced author may prefer common abbreviations and acronyms for convenience of typing and reading. When we want to make modifications to the script, readability is crucial, but readability in this context refers to how easy the script is to understand. The second popular assertion follows from this. Structured scripting is similar to structured programming in that it uses special instructions to govern how the script is executed. [19].

All test tool scripting languages, most likely, support three basic control structures. The first is called' sequence,' and it's the same as the linear scripting style we discussed before. The first command is executed first, followed by the second, and so on. ‘Selection' and 'iteration' are control structures. A script's decision-making capacity is provided via the selection control structure. An if statement examines a condition to see if it is true or false, which is the most common form. A script might need to verify that a specific message has been displayed on the screen, for example. If it has, it may proceed; if not, it must abort. The iteration control structure of a script allows it to repeat a sequence of one or more instructions as often as necessary. A 'loop' is a set of instructions that can be repeated a specific number of times or until a condition is met. If a script was needed to read data records from a file, for example, the sequence of instructions would read and process the data records [20].

Chapter 3: Proposed Research Methodology

Several To discuss research methodology we need to discuss the research process. The research process often begins with a very broad idea for a topic that would like to know more about. Some preliminary research to be done to identify a problem. After refining research questions, laying out the foundations of research design leading to a proposal that outlines the ideas and plans.

Figure 3.1: Research Process

3.1 Identify the Research Problem

The research objective refers to the primary reason for doing the study. It might be to contribute to existing area expertise, cover the research gaps, design and test a solution to a problem, and so on. In this chapter, we looked at the approaches we utilized to obtain and expand our knowledge. The thesis work's appearance and feel, as well as the procedures used to create it. This chapter covers the research methodology as well as a theoretical overview. Among the phases of this research are the efficiency of software testing and the efficient software testing. Review prior research findings, software testing approaches, testing strategies, and research-based qualitative and quantitative analyses. The first section of our paper examines the efficacy of SDLC models. We will discuss the testing techniques and also the strategies. In this technological era, technologies are developing pretty fast. To cope up with this, software companies need to start using innovative and fast-paced methods. So, we are presenting a calculative result where it shows which is the efficient software testing(manual/automation) based on our research.

3.2 Literature Review

This research is a tertiary study in the field of software testing, and it examines secondary studies in the field. A tertiary review is a systematic review of systematic reviews, often known as a ‘meta-systematic-review,' according to Kitchenham and Charters[20].

There are no tertiary programs in software testing that we could identify. However, only a few tertiary studies have been conducted in Southeast Asia. Due of research setting similarities between those works and ours, we will quickly review them next. Our search for tertiary studies in SE yielded 10 findings [20]–[28], which are arranged by year of publication in Table 1. Seven of the tertiary studies focused on the general SE rather than specific sub-areas, while two concentrated on specific sub-areas in SE: Agile practices in the global/distributed SE [29], and SLRs in the global/distributed SE [24].

Table 1. Secondary studies

#	**Title**	**Years**	**Ref.**
1	SLRs in SE – A SLR	2009	[20]
2	SLRs in SE– A tertiary study	2010	[21]
3	Critical appraisal of SLRs in SE from the perspective of the research questions	2010	[23]
4	Research synthesis in SE-A tertiary study	2011	[25]
5	Six years of SLRs in SE-An updated tertiary study	2011	[22]
6	Signs of Agile Trends in Global SE Research-A Tertiary Study	2011	[29]
7	Systematic approach for identifying relevant studies in SE	2011	[28]
8	SLRs in Distributed SE-A Tertiary Study	2012	[26]
9	A tertiary study: experiences of conducting SLRs in SE	2013	[30]
10	Risks and risk mitigation in global software development: A tertiary study	2014	[26]

3.3 Research Design

The research design is the general method we use to combine the many elements of the study in a cohesive and logical manner, ensuring that we properly solve the research topic. It also serves as the basis for data collecting, computation, and analysis. A research design's purpose is to guarantee that the evidence gathered allows us to properly address the research topic in a logical and straightforward manner.

We offer a quick overview of what we will be doing in this article in the introduction. Motivation, problem statement, aim, research outcome, and research plan are all included in the introduction. A review of the region being examined, current information around the topic, prior studies on the issue, and so on are all part of the background research for a thesis. The processes or strategies used to find, collect, evaluate, and analyze information on a topic are referred to as research methodology. The methodology portion of a research article helps readers to critically analyze the study's overall validity and dependability. The study of our research study is represented in the Discussion section. It must be presented in a way that is both intelligible and engaging to readers who are keen to know more about the outcomes analysis. The conclusion is the final section of a thesis. Results, on the other hand, are the last stage of a data processing or survey data.

The importance of software testing in ensuring software quality cannot be overstated. Following the production of the code, it must be tested to find any flaws, which must then be fixed before the software can be deployed. Although it is unrealistic to detect and debug all flaws with vital software at every level, every effort is made to eliminate as many mistakes as possible. Testing can aid in the detection of bugs in software, but it does not guarantee that it is bug-free. Software testing strategies are a means of incorporating software test case generation methods into a well-planned chain of actions that can result in effective software development. It provides the groundwork for testing. The software testing strategy should be adaptive enough to allow for the development of a custom testing approach. The software testing strategy is created by project managers, software developers, and testing professionals.

3.4 Research Analysis

We will try to give a short outcome on the efficiency of different testing techniques, as well as a short outcome on the testing strategies, whether it is acceptable or not, based on both qualitative and quantitative analysis from the research we have done. We will try to show some calculative discussion and try to find out which testing technique is more efficient[5].

Chapter 4: Discussion & Analysis

Several In the It is impossible to overestimate the importance of software testing in assuring software quality. Following the creation of the code, the program must be tested to identify any faults, which must then be debugged before the software is released. Although finding and debugging all problems in essential software at every level is impractical, every effort is made to eradicate as many errors as feasible. Testing can help find problems in software, but it doesn't ensure that it's bug-free[31]. We separated testing methods into two groups:

1. Static Testing
2. Dynamic Testing

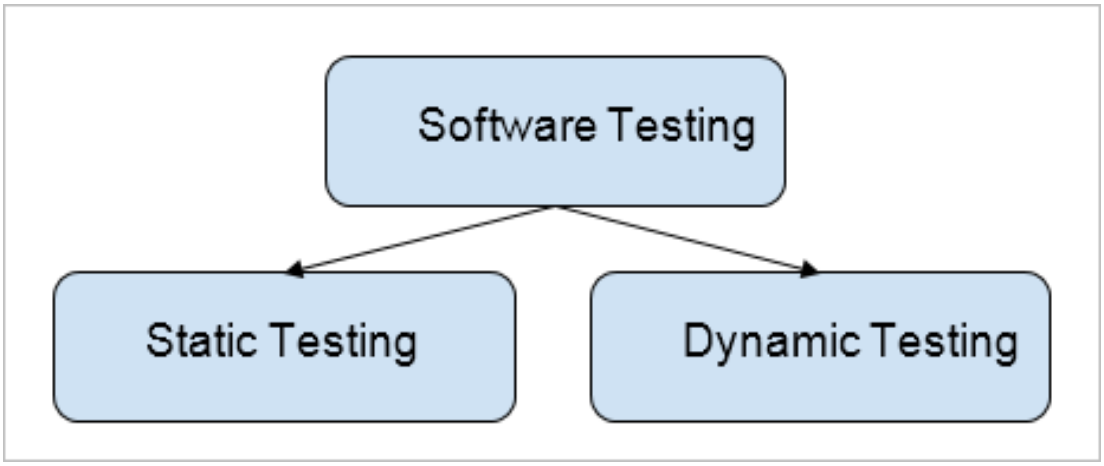

Figure 4.1: Testing Techniques

4.1 Static Testing:

Static Testing- (Manual Testing): It is a method of testing that does not need the execution of code. This stage does not entail the hiring of highly skilled employees since it does not involve the actual installation of the system. It's also known as verification testing since it starts with the first step of the software development life cycle (SDLC). The primary purpose of static testing is to improve software quality by supporting software engineers in identifying and correcting defects early in the process. Papers including such software requirements specifications (SRS),[32] technical requirements, source code, test scripts, and web page content are all subjected to static testing. It is performed out before any code is deployed. As a consequence, it is possible to examine both code and documentation. Techniques for static testing include:

- Inspection: This is mostly done to look for defects. The moderators lead the code walkthrough. Checklists are used to assess the working document in this kind of formal review.

- Walkthrough: This is not a structured procedure. This process is being led by the writers. The Author guides the attendees through the paper using his or her own thought process in order to achieve a common understanding. It's especially beneficial for papers with a greater level of complexity, such as requirement specifications.
- Technical Appraisals: A technical evaluation is performed to see if the code meets technical norms and standards, which may involve test methodologies, test methods, and test scripts.

4.2 Dynamic Testing:

Dynamic Testing- (Automated Testing): Automated testing is a kind of testing that examines a program's dynamic behavior. Validation, frequently known as automated testing, takes into account the actual system [33]. It necessitates the use of a highly skilled individual who is well-versed in the field. Dynamic testing looks at both the software's output and its input values. Progressive testing may be classified into two types:

A. White Box Testing

B. Black Box Testing

C. Grey Box Testing

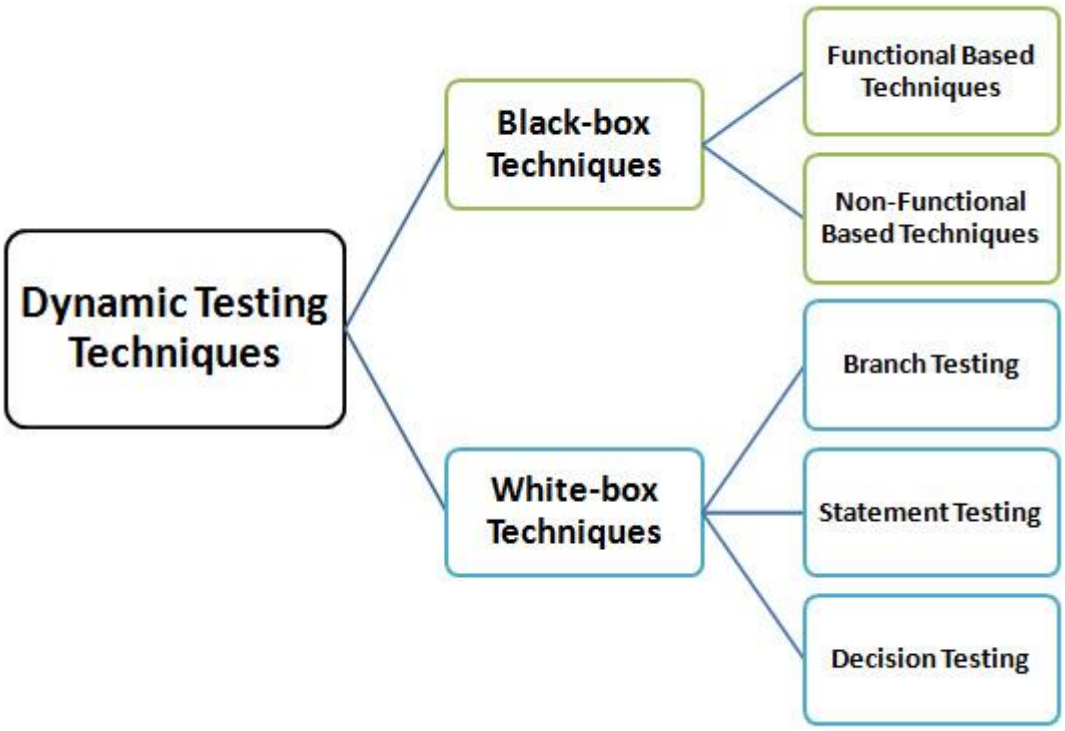

Figure 4.2: Dynamic Testing Techniques

4.2.1 White Box Testing:

The core specifications and architecture of the system are made public. As a consequence, detecting and resolving problems is incredibly cost-effective. Bugs will be discovered before a conflict starts. As a consequence, we'll refer to this method as "software testing using data from the structure and code of the program." Precision box analyses, white box analysis, structural testing, clear box testing and component testing are all terms used to describe white box testing. It's a way of finding flaws wherein the tester has complete knowledge of the situation. This approach is seldom used for troubleshooting in large systems and networks. There are several types of white box testing, including route testing, loop testing, primary control testing, and more. White-box testing investigates a program's structural features or workings by using programming talents and the system's home environment to generate test cases. The tester provides inputs so that code routes may be applied and the correct outputs can be produced. This is analogous to putting the nodes of a circuit to the test. White-box Testing may be done at the unit and as well in the test procedure. It is usually done on a unit-by-unit basis. Although this type of testing may find a lot of flaws, it can also miss needs that haven't been satisfied or specification parts which haven't been implemented [16].

White-box testing employs the following techniques:

1. API testing investigates an application's usage of public and protected APIs by building tests to ensure that particular code coverage requirements are met.

2. Fault Diagnosis Methods - Intentionally injecting defects into testing processes to measure their effectiveness.

3. Source code tools may assess the robustness of a test suite created in any way, even via black-box testing. This enables the developers to evaluate portions of such systems that are seldom tested while still ensuring that the most important functionalities are validated.

4. Function coverage is an approach that provides information about the functions which have been executed.

5. The number of lines which must be analyzed in order for the test to achieve 100% is determined by statement coverage. It ensures that every code path or branch is executed at least once. This assists in the correct running of the system.

Advantages:

1. It identifies a hidden fault in the code by deleting redundant lines of code.

2. During the design of the test outline, the greatest coverage is accomplished.

3. In a non-obtrusive way, the developer explains the reasons for implementation.

Disadvantages:

1. This procedure demands the use of a professional tester who is familiar with the inside structure.

2. Many paths will go undiscovered since it is tough to weigh all of the benefits and drawbacks.

4.2.2 Black Box Testing:

A black box testing is the one in which a consumer has no knowledge of or access to the system's internal information or workings. It meets all specifications and output criteria. The primary purpose is to determine the system's needs. There is little or no data about the system's underlying logical structure in black box testing. As a consequence, it just examines the system's basic properties. It guarantees that all inputs and outputs are correctly accepted and created. Black-box testing evaluates a system's operation without understanding how it operates. The testers just know what the software is meant to do; they don't know how it's supposed to do it. This is only in the interest of the customers. The tester only knows the set of outputs and inputs[9].

The following are examples of black-box testing methods:

1. Equivalence Partitioning divides the input domain of a program into comparable classes out of which test cases can be built. As a consequence, it is possible to minimize the set of test cases.

2. Boundary Value Analysis (BVA) is a kind of testing that focuses on the margins, or extremes, of the range value. All of the values are presented, including the minimum, maximum, error, and mean.

3. Fuzzing: This approach supplies random data to the software. It works by inserting incorrect or partly damaged data into such an automated or semi-automated session to find implementation issues.

4. Orthogonal Array Testing: This approach has a restricted input domain, but it is big enough to enable comprehensive testing.

5. The development of a graph and the assessment of the link between both the causes and their effects are the first steps in this testing approach.

6. The basic goal of all pair testing is to create a test suite which covers all potential pairings. Test cases are created to run through all of the possible discrete variations of every pair of input parameters.

7. State Transition Testing: This method of testing may be used to browse a graphical user interface.

Advantages:

1. It is not necessary for testers to be proficient in any programming languages. Testing is done from the user's standpoint.

2. It aids in the identification of any ambiguities or inconsistencies in the specified requirements.

3. Validators and developers are both self-contained entities.

Disadvantages:

1. It's hard to create test scenarios without realistic circumstances.

2. The possibility of the developer repeating previously completed tests.

3. Some components of the rear end aren't even put through their paces.

4.2.3 Grey Box Testing:

Grey box testing is a method of software assessment that does not need a thorough understanding of the application's underlying structure and design. According to the description, it's a testing software suite containing internal logic and basic code data. It makes use of internal algorithms and data structures. This strategy demands doing integration testing on two or more code modules authored by distinct developers. In this way, the value problems are established using reverse engineering. Grey box testing seems to be a quasi and unbiased testing approach [9].

Grey-box testing, that is used to build and execute tests at the user end, focuses on internal algorithms and data structures. The tester has not had complete access to the source code of the product. The following are examples of grey-box testing subtypes:

1. Stage-Model-Testing: This method verifies each object's function, transition, and transition pathways at each stage.

2. Class-Diagram Testing: This technique examines all descendant classes of the base class.

3. Testing by Sequence Diagram: This method checks each operation in the case diagrams.

4. Thread-Based Testing: This technique integrates and tests all of the types of single Use-Cases. This process is continued till all the Use-Cases courses have been properly examined.

5. Use-Based Testing: This kind of testing is carried out on programs that need or do not require the services of other programs.

Advantages:

1. It combines the benefits of black-box and white-box testing techniques.
2. The tester has the capacity to build sophisticated test scenarios in grey box testing.
3. The testing is fully unbiased.

Disadvantages:

1. The test result is restricted due to a lack of accessibility to source code.
2. Many other program options have still to be explored.
3. It's possible that a few of the test scenarios are unnecessary.

4.3 Nonfunctional Testing Techniques:

This is a form of testing used to evaluate a system's many features, including such stress, load, and etc. Nonfunctional testing is done at every stage of the testing process. It emphasizes on nonfunctional needs and is used to assess a system's readiness using criteria that aren't considered by functional testing [16].

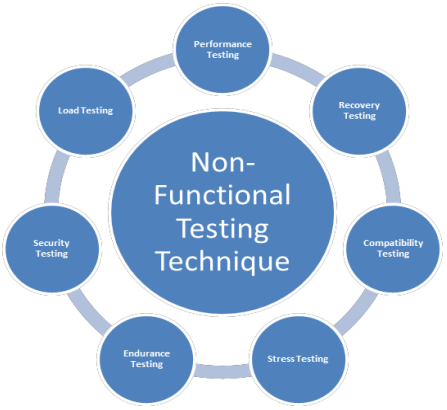

Figure 4.3: Non-functional Testing Techniques

a. Performance Testing: This evaluates the system's overall performance. Which used to see how well a system performs under a certain load.

b. Load testing: This load test is carried out to confirm that the load bearing capability of an order is met. Load testing is used to determine how well a system functions under normal and high load situations.

c. Endurance testing: It is a method of determining how longer something can endure. It's a kind of testing that looks at how the system responds after a particular period of time has elapsed. For example, a strategy may work for about the first hours, but after three hours, its efficiency starts to degrade.

d. Stress Testing: It is a technique for determining how well a system functions when it is used beyond its usual operating capabilities, often to the failure point. It's all about the load-carrying capability of the system.

e. Security Testing: Security testing is conducted to see whether a system is safe or not.

It is a method that guarantees that data is safe and so an information system continues to work as intended.

f. Recovery Testing: This is done to check if the system can be recovered after a loss or an equipment failure. The program is designed to fail in a specific situation, and then the recovery is analyzed.

g. Compatibility Testing: It determines if a software is functioning with its surroundings. It examines how effectively the completed system integrates with other components such as hardware, supplemental software, database systems, and system software, among others.

4.4 Software Testing Strategies:

Software testing strategies are an approach to implement software test case creation methods into a well-thought-out series of actions that can lead to successful software development. It lays forth the path for testing. The software testing strategy should be adaptable enough to allow for the creation of a customized testing strategy. Project managers, software developers, and testing specialists create the software testing plan. Software testing techniques are divided into four categories[34]:

1. Unit testing

2. Testing of the integration

3. Validation/acceptance testing

4. System evaluation

Unit testing:

Unit testing is the smallest testable portion, which is the smallest collection of testable lines of code. Because sufficient knowledge of fundamental programming design is required, unit testing is done by the developer. Unit testing is sometimes referred to as a white-box testing method since it focuses on evaluating code as it is developed rather than analyzing adherence to a set of criteria. Unit testing has a number of advantages [7], [8].

1. A low-cost testing procedure.

2. Simple testing approach since the code's smallest testable unit is tested here.

3. Individual pieces are evaluated as needed, without having to wait for the whole of the system to be checked.

4. Unit testing may be done in parallel by several engineers resolving bugs at the same time.

5. Defect detection and eradication are substantially less expensive than other types of testing.

6. Be able to use one of the many formal testing methodologies for unit testing that are available.

7. Simplify debugging by restricting the number of available code places in which to look for errors to a minimal number.

8. In larger integrated systems, be able to verify internal logic that is not easily touched by external inputs.

9. Achieve a high level of code structural coverage.

10. It avoids long compile-build-debug cycles while debugging serious problems. Techniques for unit testing include: Several successful testing methodologies are used in unit testing. There are three sorts of testing techniques:

i. Structural Testing

ii. Intuitive or Heuristic Testing

Integration Testing:

Integration testing is a good way to build a software structure and test for interface concerns at the same time. The purpose of integration testing is to bring all of the unit-tested components together and test them all at once. Top-down integration testing and bottom-up integration testing are two different types of integration testing strategies[31].

a. Top-down Integration: This is a walk guide on putting together a program format. Formats are integrated downward through the fabric, starting with the principal control module. The subordinate modules of the centralized module are incorporated into the framework in either a detail or a breadth-first manner. There are five stages to the integration process:

1. Stubs are utilized to replace all components directly subordinated to the main control module, and the principal control module is being used as a test driver.

2. Select subordinate drafts are substituted one at a period with actual parts, depending also on integration methods.

3. As each element is incorporated, tests are carried out.

4. The genuine element is substituted with a fresh stub when each round of trials is completed.

5. Regression testing can be controlled to prevent the introduction of new faults. It's not quite as simple as it seems.

In this case, it's probable that a logistical difficulty may arise. The issue emerges when analyzing a low-level component that demands examination of the higher level. Stub replaces low-level modules at the beginning of top-down testing. As a consequence, no data may increase in value.

b. Bottom-up Integration: To begin the testing and development process, atomic modules are employed. Because modules are connected from the bottom up, processing for things subordinate to a required level is always available. The steps of a bottom-up integration approach are as follows:

1. To perform a certain software function, low-level elements are clustered together into clusters.

2.The coordination of test case outputs and inputs is entrusted to a driver.

3. The members of the group are put to test.

4. Drivers are separated and clusters are merged also as program structure evolves.

Acceptance Testing:

This kind of testing is used to guarantee that the product is constructed in accordance with industry standards and extensive requirements, and that it meets all the user's requirements. When a product is manufactured by a third party, the user does this kind of testing. Acceptance testing is a kind of black-box testing in which the users are not engaged in the internal workings of the system. Acceptance testing is also known as validation testing, quality assurance testing, final testing, manufacturing acceptance testing, and software testing. In software engineering, acceptance testing can be done at two levels: at the service provider level and also at the end-user level. There are four different types of acceptance testing[7].

1. User Acceptability Testing: Before the system is delivered to the end user, it must pass user acceptance testing. User acceptability testing may be conducted by the program itself to guarantee that it can do the required job in real-world circumstances.
2. Alpha and Beta Testing: This form of testing is usually handled by QA teams or developers. A mixture of unit, integration, including system testing is referred to as "Alpha testing." With or without the participation of developers, alpha testing is carried out. The following features are reviewed throughout the testing: spelling problems, broken lines, hazy directions, and so on. Beta testing starts once alpha testing has completed successfully. Beta testing is done by actual users who are using the app in real-world circumstances. Customers submit feedback to the developer on the experiment's findings. Field testing is another name for it. Before the system/product is made accessible to more users/customers, user feedback is used to improve productivity.
3. Operational Acceptance Testing (Functional Testing): This kind of testing guarantees that the system's required processes and procedures are all in place and that the user/tester can utilize it.
4. Contract Acceptance Testing: The system is put to test against the contract's established criteria, as well as to see whether it complies with all relevant federal, state, and local laws and regulations, as well as all significant requirements.

System Testing:

This is a kind of hardware or software testing which includes evaluating the conformance of a full, integrated system to its set of criteria. The term "black-box testing" refers to the testing of systems. The goal of testing is to determine whether the system meets the requirements. A few instances of various forms of system testing are as follows:

i. Testing for recovery

ii. Testing for security

iii. Testing of graphical user interfaces

iv. Testing for compatibility

1. An application's ability to recover after a crash, hardware failure, or other issue is tested using recovery testing. If the process is automated, it is important to ensure that all of the steps in the recovery process are checked for accuracy. Recovery testing makes the program fail in a number of ways to make sure that it is rebuilt correctly.
2. Security testing is performed to determine if a system's built-in defensive measures are able to prevent unwanted entry. In security testing, each tester has a personal motivation to get into the system. Any number of techniques, including clerical, bespoke software, or system overload, may be employed in an attempt by the validator to get access to the system and refuse service to other users. The purpose of security testing is to find and analyze any possible holes or threats to the system.
3. Testing the product's graphical user interface to ensure that it fulfills set standards is called graphical user interface testing. Toolbars and menu bars, conversation boxes, and windows are all examples of GUI panels that may be used for GUI testing.
4. Testing for interoperability examines how well a system interacts with its environment. It checks, among other things, how well the final system works with hardware, software, database management systems, and operating systems from third parties.

4.5 Manual and Automated Testing:

When the program is tested by hand without the use of any tools then it is considered by manual testing[5].

Automation testing is often referred to automation testing. The tester creates scripts and tests the program using another piece of software or a tool. It is essentially a procedure that automates a manual process. The following table 2 illustrates the distinction between manual and automated testing:

Table 1. Comparison between manual testing and automation testing

	Manual Testing	**Automation Testing**
1	Manual testing takes a long time to complete since it is carried out by human resources. So, it is time-consuming and exhausting	Automation testing completes test cases far more quickly than human resources.
2	Automation testing completes test cases far more quickly than human resources.	Because test cases are executed using automation technologies, automation testing requires fewer testers. As a result, there is a reduction in investment in the field of human resources.
3	No programming can be done in manual testing to construct elaborate tests that retrieve the secret data.	Automation testing is programmable, which means that testers may create complex tests to uncover hidden issues.
4	Manual testing is less trustworthy since, due to human error, tests may not be done with accuracy every time.	Automation testing is more accurate and less prone to errors.

There are several additional forms of software testing; in fact, every sort of software testing may be performed manually or with the help of an automated tool. Unit, Integration, System, Load testing and other software testing categories fall within the aforementioned two testing approaches[5]. As seen in figure 4 below.

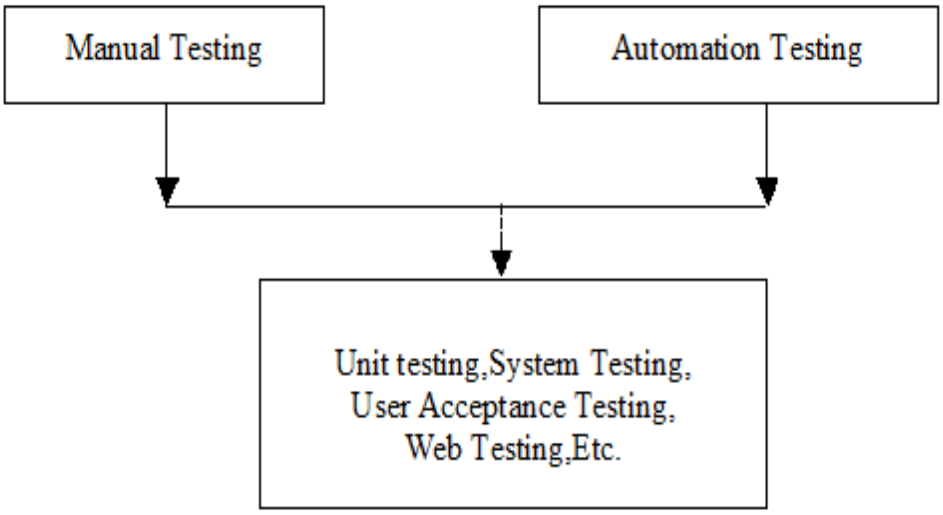

Figure 4.5: Manual testing & Automation testing flow

This section compares and contrasts the advantages of employing computerized scripts against manual testing. To this purpose, Daniel L Asfaw's study includes three test scenarios[35]. The first two aid in efficiency comparison by examining the best test time attained by both human and automated meanness. The last test case is designed to assess accuracy by counting the number of times testers were able to accurately count the links on the Indian Express site out of 100 tries.

4.5.1 Efficiency: Automated testing vs manual testing:

Table 2: Test cases

Test case	Manual best test time	Automation test time
Test case 1	300 seconds	20 seconds
Test case 2	459 seconds	30 seconds

That is best for running test case one was 300 seconds, but the automated script was able to do it in 20 seconds as seen in Table 3, the traditional testers' time. There is a difference that is 280-second them. The time that is best recorded for test cases two and three by hand testers was 459 seconds, and 30 seconds via automation in a similar vein. The difference is 429 seconds once more. In the period spent executing these test cases, we notice pairs of relationships (manual verses automation)[9]. We might construct a liner equation with the slope that is following on these order pairs:

$$N = \frac{(y1 - y2)}{(x1 - x2)}$$

Where m is the slope and yi is the time spent manually testing the test case. xi = time spent automating testing for the test case[5].

$$N = \frac{(459 - 300)}{30 - 20)}$$
$$N \equiv 15.9$$
$$y - y1 \equiv N(x - x1)$$
$$y - 300 = 15.9(x - 20)$$
$$y = (15.9x - 318) + 300$$
$$y = 15.9x - 18$$

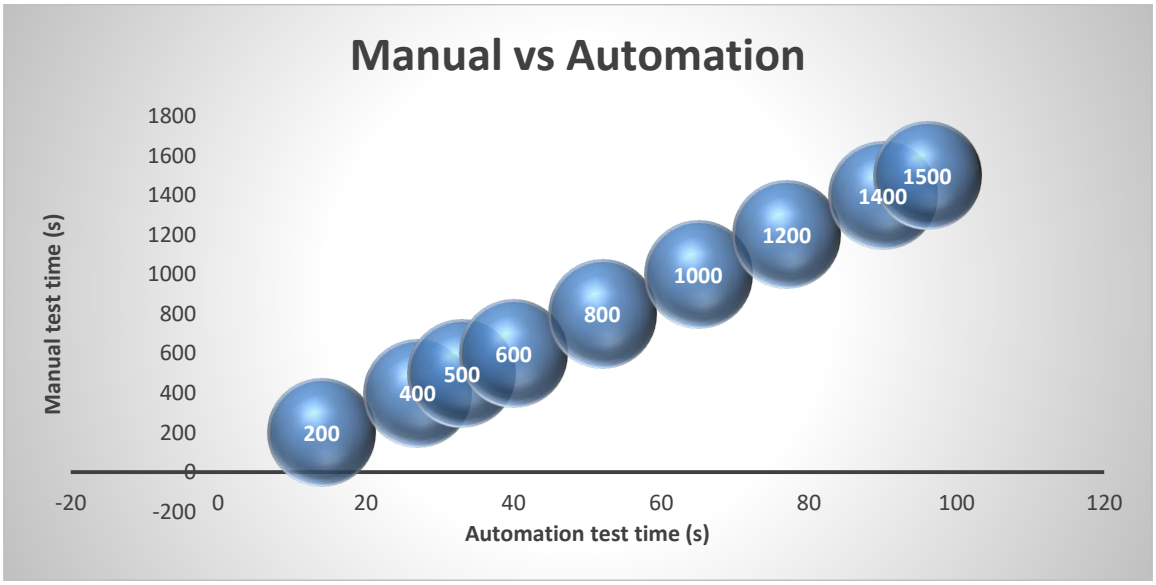

Figure 4.5.1: Manual vs Automation testing time graph

An automated test takes seconds to complete, but a manual test takes longer, as seen in Figure 5. Using an automated test case, a test case that would take 1000 seconds in human testing only takes 65 seconds, according to this graph. A test that took 500 seconds to do manually only took 33 seconds to run automatically as well. Even when we approach zero points on the graph, the advantage of automated testing diminishes rapidly. This indicates the time required to run a manual test case.

Automated scripts, even if they take longer to implement, are more beneficial and lucrative since they provide greater timing and efficiency.

Automated testing would take only 3600 seconds to complete a test case that would take the same amount of time if done manually (158.95 hours) (1 hr). This is only going to happen once.

If this test case were to run every week for a year, what would the results be?

It will take 8265.4 hours (2975544 seconds) to do it manually, and 52 hours of automation to do it automatically. Thus, the time saved by using automated test scripts would be 8213.44 hrs. as a result. This would arithmetically develop to a significant number of hours if anticipated for ten years.

This does not mean that the investment will be equivalent to the number of hours saved by increasing efficiency. These differences are due to a lack of equality between manual and automated testers on an employment level. Automated testers are compensated slightly better than manual testers on the market. Manual testers make "X" dollars per hour, but automation testers make "2X" dollars per hour, hence in this instance they would make 8265.4X dollars per year, while the automated testers may make 104X dollars per year. As a result, an automated test case would save 8101X the amount of money. When a test case is not repetitious, time-consuming, or labor-intensive, manual testing may be a better alternative than automated scripts[5].

Consequently, automated scripts are suggested over manual regression testing for a number of reasons, including the following: It's important to note that regression testing involves a lot of work. It occurs on a regular basis. As a second point, it requires a large number of tests. Third, it is time-consuming and labor-intensive. Finally, it's pricey to do so.

To make it more time-saving, less labor intensive, and cost-effective to use automated scripts. When all is said and done, the investment would pay off handsomely.

4.5.2 Accuracy: Automated testing vs manual testing:

If a set of measurements is compared to a known standard, the computing accuracy might tell us how near we are (Samuel S., Computer Sciences, 2014)[35][7]. The vast majority of those he spoke with on the subject said that figuring out correctness was a huge pain. His experience as an automation engineer had taught him to be wary of relying on automated scripts to pass tests that had previously been passed by manual testers.

There might be a number of causes for this. It is first and foremost difficult to test or manually handle the criteria. Manual testers are also error-pruned in difficult and complicated test scenarios, like as counting the number of links on a web page.

This research solely looked at accuracy. This condition must be met by keeping track of the number of links on the newsletter's home page.

As a result, a test case has been created to handle this requirement's testing and verification points. This exam will need 50 manual testers to complete. A total of 100 tries must be made in order to calculate the percentage of right attempts. Manual testers have been grouped into five categories based on their level of experience. No matter how many times automated scripts run the identical test scenario, the results are the same[9].

There is a second table at this location It's possible that "A" and "B" can be used to represent the group's average years of experience, while "C" and "D" can be used to represent the group's average number of correct responses out of 100 tries, respectively.

The output of automation is shown as a single unit in the second row from the bottom. whereby the second row from the bottom displays the automation output as a single unit of analysis[35].

Table 3: Accuracy table

A	B	C	D
2.916	1309.58	24.24	1439.81
4	1581.3	23.152	1168.14
5.571	1809.3	21.6	940.14
20.42	2853.8	-20.421	-2853.8
30	7848	-30	-7847.5
100	1095	-100	-1095
Mean of A=27.152	Mean of B=2749.4		-7153.2

Table 4 shows that out of 100 attempts, testers with little or no connection to academic skills and testers with 3 to 5 years of work experience earned a mean score of out of 100 tries. On an average of 5.57 out of 100 trials, participants in the final three groups,

each with a unique set of skills derived from years of work experience, received the correct answer. The quality of one's work improves as one's academic importance increases. That is to say, just because something is relevant to academia does not mean it is right. However, the results of the test show that the correct and expected reaction has been received in every case. Hand testing may not be as accurate as automated scripts, which might lead to extra study.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

The primary objective of every project is to provide a high-quality software product. Because of this, testing is an extremely vital part of the development process for any type of software. Many tools for testing have been created and are currently available on the market. It became imperative for the testing team to learn about various software testing technologies in order to compete with other software developers and offer a high-quality product on the market. Research on automated testing and manual testing in different platforms is presented in this article, highlighting the importance of automation over manual testing and concluding that there is a need to understand the testing demands and create a program to verify those needs.

Manual testing takes time, is laborious, and necessitates a significant investment in human resources. We can use automation tools to capture the test suite and replay it if necessary. There is no need for human interaction after the test suite has been automated. The initial costs in automation testing are more than in manual testing, and you can't automate everything. Determine which test cases (human or automated) will deliver the best return on investment. Metrics are critical for assessing the quality and success of both automated and human software testing efforts. The test metrics provide visibility into the product's readiness and provide a clear evaluation of the product's quality and completeness. To analyze the efficacy of both methods of testing, this article covered several manual testing measures and automation metrics such as test execution, test case productivity, defect rejection metrics, and test coverage metrics.

5.2 Limitations

We discussed about automation testing and manual testing in our paper. We also compare efficient software testing for assuring software quality and which one is best and cost efficient for industry. But in this paper, we did not discuss about the tools of automation and compare them with each other to ensure the best efficient automation tools for assuring software quality.

5.3 Future Work

This research examined into the various approaches to automation testing and explored the various open-source testing tools available. It reported the findings of the study based on available literature and current trends. This research can be expanded and made more effective by incorporating practical performance analysis on the open-source tools available for automation testing, so that both researchers and industry professionals can gain a better understanding of purpose-specific approaches and tools.

References

[1] C. Klammer and R. Ramler, “A Journey from Manual Testing to Automated Test Generation in an Industry Project,” *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. Companion, QRS-C 2017*, pp. 591–592, 2017, doi: 10.1109/QRS-C.2017.108.

[2] M. Polo, P. Reales, M. Piattini, and C. Ebert, “Test automation,” *IEEE Softw.*, vol. 30, no. 1, pp. 84–89, 2013, doi: 10.1109/MS.2013.15.

[3] L. Osterweil, “Strategic directions in software quality,” *ACM Comput. Surv.*, vol. 28, no. 4, pp. 738–750, 1996, doi: 10.1145/242223.242288.

[4] D. Galin, “Organization for Assuring Software Quality,” *Softw. Qual. Concepts Pract.*, pp. 58–80, 2018, doi: 10.1002/9781119134527.ch4.

[5] M. Sharma and R. Angmo, “Web based Automation Testing and Tools,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 908–912, 2014.

[6] T. Wissink and C. Amaro, “Successful test automation for software maintenance,” *IEEE Int. Conf. Softw. Maintenance, ICSM*, pp. 265–266, 2006, doi: 10.1109/ICSM.2006.63.

[7] M. Gligoric, S. Negara, O. Legunsen, and D. Marinov, “An empirical evaluation and comparison of manual and automated test selection,” *ASE 2014 - Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng.*, pp. 361–371, 2014, doi: 10.1145/2642937.2643019.

[8] K. Sneha and G. M. Malle, “Assistant Professor in Computer Science Department,” *2017 Int. Conf. Energy, Commun. Data Anal. Soft Comput.*, pp. 77–81, 2017.

[9] R. M. Sharma, “Quantitative Analysis of Automation and Manual Testing,” *Int. J. Eng. Innov. Technol.*, vol. 4, no. 1, pp. 252–257, 2014.

[10] R. Ramler, S. Biffl, and P. Grünbacher, “Value-based management of software testing,” *Value-Based Softw. Eng.*, pp. 225–244, 2006, doi: 10.1007/3-540-29263-2_11.

[11] D. S. Rosenblum and E. J. Weyuker, “Lessons Learned from a Regression Testing Case Study,” *Empir. Softw. Eng.*, vol. 2, no. 2, pp. 188–191, 1997, doi: 10.1023/A:1009717821137.

[12] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, “Study of effective regression testing in practice,” *Proc. Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 264–274, 1997, doi: 10.1109/issre.1997.630875.

[13] L. Nagowah and P. Roopnah, “AsT-A simple automated system testing tool,” *Proc. - 2010 3rd IEEE Int. Conf. Comput. Sci. Inf. Technol. ICCSIT 2010*, vol. 9, pp. 301–306, 2010, doi: 10.1109/ICCSIT.2010.5563986.

[14] R. J. Lipton and F. G. Sayward, “Hints on test data selection: Help for the practicing programmer,” *Computer (Long. Beach. Calif).*, vol. 11, no. 4, pp. 34–41, 1978, doi: 10.1109/C-M.1978.218136.

[15] G. Rothermel, R. H. Untch, and M. J. Harrold, “Test Case Prioritization 1 Introduction,” *Test*, no. December, pp. 1–32, 1999.

[16] G. Myers, “软件测试的艺术(第二版),” *Softw. Testing, Verif. Reliab.*, vol. 15, p. 234, 2004, [Online]. Available: http://www.noqualityinside.com/nqi/nqifiles/The Art of Software Testing - Second Edition.pdf

[17] T. Varma, “Automated software testing,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 25, no. 3, pp. 65–65, 2000, doi: 10.1145/505863.505890.

[18] J. Hartmann and D. J. Robson, “Techniques for Selective Revalidation,” *IEEE Softw.*, vol. 7, no. 1, pp. 31–36, 1990, doi: 10.1109/52.43047.

[19] E. J. Weyuker, “The evaluation of program-based software test data adequacy criteria,” *Commun. ACM*, vol. 31, no. 6, pp. 668–675, 1988, doi: 10.1145/62959.62963.

[20] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering - A systematic literature review,” *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009, doi: 10.1016/j.infsof.2008.09.009.

[21] B. Kitchenham *et al.*, “Systematic literature reviews in software engineering-A tertiary study,” *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, 2010, doi: 10.1016/j.infsof.2010.03.006.

[22] F. Q. B. Da Silva, A. L. M. Santos, S. Soares, A. C. C. Frana, C. V. F. Monteiro, and F. F. MacIel, “Six years of systematic literature reviews in software engineering: An updated tertiary study,” *Inf. Softw. Technol.*, vol. 53, no. 9, pp. 899–913, 2011, doi: 10.1016/j.infsof.2011.04.004.

[23] F. Q. B. Da Silva, A. L. M. Santos, S. C. B. Soares, A. C. C. França, and C. V. F. Monteiro, “A critical appraisal of systematic reviews in software engineering from the perspective of the research questions asked in the reviews,” *ESEM 2010 - Proc. 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas.*, pp. 1–4, 2010, doi: 10.1145/1852786.1852830.

[24] S. Imtiaz, M. Bano, N. Ikram, and M. Niazi, “A tertiary study: Experiences of conducting systematic literature reviews in software engineering,” *ACM Int. Conf. Proceeding Ser.*, pp. 177–182, 2013, doi: 10.1145/2460999.2461025.

[25] D. S. Cruzes and T. Dyb, “Research synthesis in software engineering: A tertiary study,” *Inf. Softw. Technol.*, vol. 53, no. 5, pp. 440–455, 2011, doi: 10.1016/j.infsof.2011.01.004.

[26] J. M. Verner, O. P. Brereton, B. A. Kitchenham, M. Turner, and M. Niazi, “Risks and risk mitigation in global software development: A tertiary study,” *Inf. Softw. Technol.*, vol. 56, no. 1, pp. 54–78, 2014, doi: 10.1016/j.infsof.2013.06.005.

[27] T. MAREW, J. KIM, and D. H. BAE, “Systematic Mapping Studies in Software,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 17, no. 1, pp. 33–55, 2007, [Online]. Available: http://content.ebscohost.com/ContentServer.asp?T=P&P=AN&K=22674743&S=R&D=bth&EbscoContent=dGJyMNHX8kSeqK44zdnyOLCmr0qeprZSr6e4SrCWxWXS&ContentCustomer=dGJyMPGosk+xq65QuePfgeyx44Dt6fIA%5Cnhttp://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=2447601

[28] H. Zhang, M. A. Babar, and P. Tell, “Identifying relevant studies in software engineering,” *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 625–637, 2011, doi: 10.1016/j.infsof.2010.12.010.

[29] G. K. Hanssen, D. Šmite, and N. B. Moe, “Signs of agile trends in global software engineering research: A tertiary study,” *Proc. - 2011 6th IEEE Int. Conf. Glob. Softw. Eng. Work. ICGSE Work. 2011*, pp. 17–23, 2011, doi: 10.1109/ICGSE-W.2011.12.

[30] S. Imtiaz, M. Bano, N. Ikram, and M. Niazi, “A tertiary study: Experiences of conducting systematic literature reviews in software engineering,” *ACM Int. Conf. Proceeding Ser.*, no. March 2015, pp. 177–182, 2013, doi: 10.1145/2460999.2461025.

[31] A. Jangra, G. Singh, J. Singh, and R. Verma, “Exploring Testing Strategies,” *Int. J. Inf. Technol. Knowl. Manag.*, vol. 4, no. 1, pp. 297–299, 2011.

[32] G. Kumar, “SK International Journal of Multidisciplinary Research Hub,” *Int. J. Multidiscip. Res. Hub J. all Subj.*, vol. 2, no. 9, pp. 2394–3122, 2015.

[33] N. Anwar and S. Kar, “Review Paper on Various Software Testing Techniques & Strategies,” *Glob. J. Comput. Sci. Technol.*, vol. 19, no. September, pp. 43–49, 2019, doi: 10.34257/gjcstcvol19is2pg43.

[34] D. Suster *et al.*, “Myxoinflammatory fibroblastic sarcoma: an immunohistochemical and molecular genetic study of 73 cases,” *Modern Pathology*, vol. 33, no. 12. pp. 2520–2533, 2020. doi: 10.1038/s41379-020-0580-6.

[35] D. L. Asfaw, “Inter national Journal of Innovative Resear ch in Infor mation Secur ity (IJIRIS) Benefits of Automated Testing Over Manual Testing,” 2015.