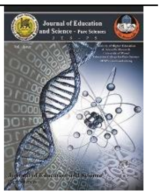
Journal of Education and Science (ISSN 1812-125X)

www.edusj.mosuljournals.com



Software Testing Techniques and Tools: A Review

S. K. Izzat*[1] N. N. Saleem[2]

Department of Software, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq

Article information

Article history:
Received: December 31, 2022
Accepted: March 06, 2023
Available online: June 01, 2023

Keywords:
White Box Testing
Path Testing
Control Flow Graph.

Correspondence:
S. K. Izzat
sakina.21csp13@student.uomosul.edu.iq

Abstract

The software development process is closely related to the creation and evaluation processes. The problem with this software development is that it often lacks testing which leads to software failures. In order to maintain a high-quality product in excellent performance condition, testing becomes critical. The software can be tested by using White Box, Black-Box, or gray testing techniques. In this investigation, the types of tests were reviewed. Performing Testing with White Box Testing uses a number of testing methodologies based on path testing, including the production of flowcharts, cyclomatic complexity assessment, and independent path testing. As a result, it is possible to implement a foundation path testing technique and white box approach to testing. This review included several axes, namely the definition of white box testing tools, then the testing techniques in general, the benefits and gains of each of these technologies, the levels of testing, and finally the steps of conducting the test. This review then came to several conclusions that are mentioned at the end of this paper.

DOI:10.33899/edusj.2023.137480.1305, ©Authors, 2023, College of Education for Pure Sciences, University of Mosul.

1. Introduction

The two areas of software engineering are: A software as a tool and product. Designing, analyzing systems, coding, testing, and performance are some of the stages that must occur within the Systems Development Life Cycle (SDLC). With a strong planning phase, aspects of systems development, coding, and testing that ensure the analysis and coding phases are consistent with one another to produce correctly built and used software, the total SDLC process has significant and interconnected processes from one to another [1].

Some currently available methods can be used in the testing process when using the SDLC, and they can be used to test both formation and object-oriented programing (OOP) software. In application testing, some potential methodologies that might be employed include White, Black, and Grey Box testing. System analysis examination known as "White Box testing" is used to spot discrepancies between system requirements and built or existing systems [2].

The examining process should be based on user needs in order to run properly and efficiently, among other things. There is a certain amount of examining time and resources; Additionally, resources for examining must be spent wisely, and should begin with basic terms and move to much more complex ones; A different panel of investigators or an outside panel must administer the examination. Additionally, the needs of the client should inform all examining; The procedure for examining software must be implemented as soon as possible in the software construction procedure as is practical, with a focus on object identification and regular updates to the original examining strategy [3].

The quality of the test itself should not be undervalued; there are a number of requirements for putting effective examining into operation, such as covering all conceivable uses of the software in the scope of examining, developing the scope of the pathways in accordance with the program framework, and not being unnecessarily simple or complex [4].

Control flow, branch, base path, data flow, and loop are all significant steps or methods in the testing process when employing the White Box approach. Meanwhile, Pressman asserts that the White Box methodology includes a number of examining approaches, including base path and control structure , which also include examining methodologies like condition, data flow, and loop testing [5].

Numerous publications on software testing methodologies have been provided in the literature over the years. The important works among them are reviewed in this area, with a concentration on those that chronologically explain all three of the main software testing approaches or just one of them in great depth. Their common weaknesses are recognized. and Table (1) below compares the evaluated literature.

Table 1. Related works Summary

Reference	Dynamic Static	Testing Level	White box	Black box	Grey box	Experienced-base	Contrastings
[6]	No	No	Yes	Yes	No	No	No
[7]	No	No	Yes	Yes	Yes	No	Yes
[4]	No	Yes	Yes	Yes	No	No	No
[8]	No	No	Yes	Yes	No	No	Yes
[9]	No	No	Yes	Yes	Yes	No	No
[10]	No	Yes	Yes	Yes	Yes	No	No
[1]	No	No	Yes	No	No	No	No
[11]	Yes	Yes	Yes	Yes	Yes	No	Yes

The fact that no papers addressed the experienced-based testing approach might be considered as a fault in the literature review of all the publications that were looked into. The majority of the literature that is currently available focuses primarily on white-box and black-box dynamic techniques; none of them examined all dynamic testing techniques or provided a clear classification order of the various testing types under each technique with a thorough explanation of some of the testing types that are most frequently used in each technique. This research aims to meet that requirement.

2. White Box Testing Tools

Given the aspects to consider before choosing a tool, choosing the perfect tool for software testing can sometimes be difficult. One of the key elements in successful test automation is the choice of test tool. This calls for research within the scope of examining and examining methodology and then selecting the appropriate examining instrument that conflicts with the requirements of an automated test suite for a particular product and version [12].

A testing tool may be used to test desktop applications, mobile applications, web applications, combinations of two applications, regression tests, integration tests, and other testing features. The tools presented below were chosen using inclusion and exclusion criteria that took into account both the frequently used by business experts and the most often discussed tools from the literature. This article includes a brief overview of these instruments as well as a tabular contrast of their pros and cons based on standards from the literature, such as cost, reliability and reusability.

A. Selenium.

Selenium is a free open source testing tool used for web application examination. It works equally well in all popular browsers and operating systems, including Windows, Mac, and Linux. Selenium encourages in a variety of programming languages, including Java, PHP, C#, Python, Groovy, Ruby, and Perl. Selenium IDE, Selenium RC, Web Driver, and Selenium Grid are the four essential parts of the Selenium suite. It is made to encourage and promote automated of the functional parts of online applications. It can also be used for the web application's black box.

Selenium is portable with all platforms and doesn't require learning new languages. it's easy to integrate with various platforms. Parallel testing with Selenium and through the cloud allow testers to receive feedback much faster and work on the changes instead of waiting overnight for a test pass. Selenium also Supports mobile testing or web mobile applications [13][14].

B. Test Complete.

Smart Bear Software developed the automated functional testing solution called Test Complete. It supports a variety of examining techniques, including GUI examining, functional examining, and unit testing. Making catchphrases that are used in testing is visual, easy, and doesn't require programming knowledge. Although scripting necessitates the understanding of the scripting instructions, it allows the tester to create more powerful and flexible tests [15]. In contrast to Selenium, Test Complete is not open source; after a free 30-day trial, it requires a license to be used.

C. Ranorex.

Renorax is an inspecting device for programmed testing that is straightforward, exhaustive, and efficient. It inspects applications according to the viewpoint of a client, making it a better option than existing instruments[20, 22]: reusable test programs, mix with various instruments, GUI acknowledgment, record and playback, bug distinguishing proof, and so forth. It

gives an additional ability to perform relapse and guarantees that test exercises might be reused. It likewise does stage similarity testing to guarantee excellent programming. Any testing group or association can use Ranorex in light of the fact that it is easy to utilize, reasonable, and accessible.
Ranorex is a white box testing automation tool, such that is used for everyone to record automation tests step-by-step for desktop, web and mobile applications. The main component of this tool includes the Ranorex Recorder, object repository, Ranorex Spy, code editor, and debugger in a single environment. Its framework is built in a manner that supports standard programming languages that help in editing recordings or creating custom tests. that help you understand the process of creating, recording, and analyzing your automated test series with ease. Ranorex also provides a detailed test summary report at the end of each and every test runs on its platform. It even provides screenshots for validation. Their reports are comprehensive, and they collaborate previous test runs with current ones. The reports can produce in PDF format[14].

D.Appium.
With its promise of efficient, bug-free, and high-quality apps, Appium delivers a completely new revolution in automation testing. This approach helps projects save time, money, and labor. Developers can create tests for many platforms, including Ios and Android, using the open source, cross-platform Appium tool. The three primary parts of it are the Appium server, Inspector, and Doctor. Since Appium supports a wide range of languages, including Java, Python, Ruby, JavaScript, and others, developers of all skill levels may utilize the tool to its full potential. Appium is mostly used to assess how well users interact with the content of mobile web applications. Test results are used to assess the accuracy, user engagement, usability, and feature availability of the mobile application [10].

E.Quick Test Professional.
Hewlett Packard created QTP (Quick Test Professional), a Windows-based programming tool for testing desktop and web applications. It supports automation for regression and functional testing (HP). To execute scripts and fill out test forms, QTP uses Visual Basic Script. It can also operate with a variety of objects and manage application testing [14]. Regression and functional testing can be automated with QTP, which takes into account every key software implementation and environment. Although Quick Test Professional is typically used for UI-based test case automation, other types of test cases, like database testing and file system operations, can also be automated [13][12].

F.Open Script.
Open Script may be used to write test scripts for a number of reasons. Numerous browsers, including Google Chrome, Mozilla Firefox, and Internet Explorer, may have their functionality recorded by the program. There are no capacity limitations because the open script was built on top of the Eclipse platform and uses Java as its scripting language. High states can also be achieved via it. It includes extremely user-friendly UI features that make it simple for those who aren't software engineers to use. Applications created with VB, Dot Net, VC++, etc. cannot be automated using Open script. [16].

G.Janova.
An automated software testing system that runs safely on the cloud is called Janova. This tool does not involve the writing of any scripts; instead, it makes use of straightforward English-based tools to streamline the process of implementing software. Because of this, both programmers and non-programmers can use it with ease. No such program is available for download; hence no infrastructure project is necessary. Its pace of execution is also faster than the conventional web testing tools because it uses the cloud and has a quick and easy setup that requires no installation. Although it is not open source, the fee for the license is reasonable [17].

H.Rational Functional Tester.
IBM developed (RFT), an automated tool based on object-oriented programming that does regression, functional, GUI, and data-driven testing. It supports Web-based Java and.NET applications created using Microsoft Visual Studio. RFT detects areas of application that were and were not performed during the examination, as well as ensuring that test plans and cases are maintained and carried out correctly by the Quality Assurance departments of businesses [17].
The type of application that needs to be tested, the project's complexity, the cost of the tool to be used, and the organization's budget for the stage all go into the decision of which automated testing tools to utilize.
Table 2. provides a full description of each tool based on a number of factors. Practitioners in the field can use this description to choose the testing instrument with confidence. The tools discussed and the requirements each tool satisfies are listed in the table. Whether it's a huge or little project, this research will assist industry professionals in choosing the most necessary equipment for software testing. Table 3. provides a thorough breakdown of these tools along with the testing types that each one offers.

Table 2. Software testing Tools factors descriptor

Tool	Open Source	License	Approving Platform (Mobile)	Approving Platform (web)	Approving Platform (Desktop)	Learning Ease/ Ease of use	Coding/ Programing skills	Code Reusability	Test Result report	Record & Playbook
Selenium	**Yes**			**Yes**		**Yes**	**Yes**	**Yes**	**Plug in**	**Yes**
Test Complete		Yes	Yes	Yes	Yes	Yes	No		Yes	Yes
Ranorex		Yes	Yes	Yes	Yes			Yes		Yes
Appium	Yes		Yes					Yes		Yes
QTP		Yes		Yes	Yes		No	Yes	Yes	
OpenScript		Yes		Yes	Yes	No		Yes	Yes	Yes
Janova		Yes		Cloud base			No		Yes	Yes
RFT	Yes			Yes		Yes				

Table 3. Tools Testing Type

Tool	Testing Type
Selenium	Functional testing
TestComplete	**Functional testing, Graphical User Interface testing, Unit testing**
Ranorex	Graphical User Interface testing, Compatibility testing
Appium	Graphical User Interface testing, Functional testing
Quick Test Professional	Functional testing, Regression testing
OpenScript	Functional testing, Load testing, Database testing
Janova	Functional testing
Rational Functional Tester	Functional testing, Regression testing, Graphical User Interface testing

3. Techniques of Software Testing

To allow for sufficient testing, there are many different technologies, each with specific capabilities. Functional specifications that promote the production of high-quality, error-free software are what determine whether two or more of these technologies should be combined. Fundamentally, there are three categories of dynamic testing techniques: Black Box, White Box, and Gray Box [18] see Figure 1.

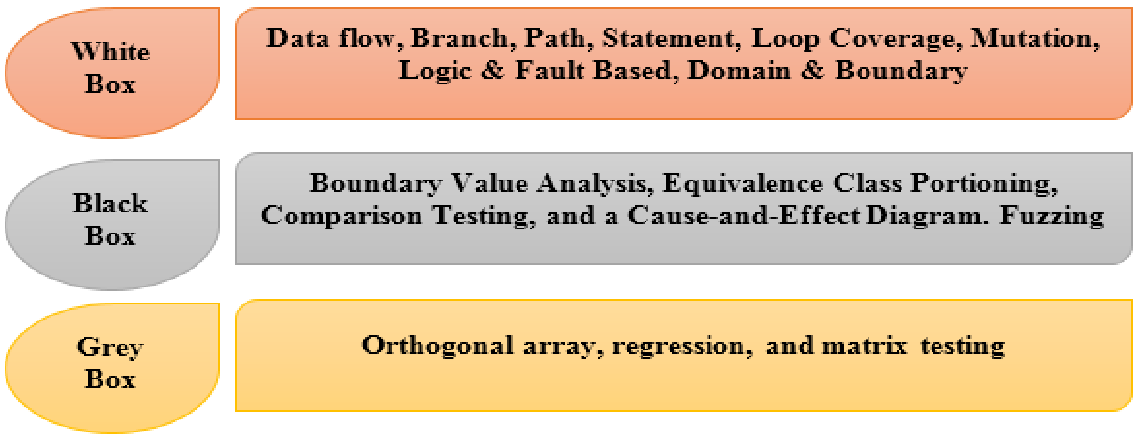

White Box
Data flow, Branch, Path, Statement, Loop Coverage, Mutation, Logic & Fault Based, Domain & Boundary
Black Box
Boundary Value Analysis, Equivalence Class Portioning, Comparison Testing, and a Cause-and-Effect Diagram. Fuzzing
Grey Box
Orthogonal array, regression, and matrix testing

Figure 1. Dynamic Testing Techniques

The complete testing process may be divided into two categories: static and dynamic testing. Both categories are complementary to one another since they both have the propensity to identify errors and flaws quickly and effectively. Static is referred to as "verification activities," whereas dynamic is referred to as "testing." A contrast of the testing methods is shown in Table 4. [52]

Table 4. Static VS Dynamic Testing

Criteria	**Static Testing**	**Dynamic Testing**
Execution	No execution required	**Required code execution**
Test case	No test case is used	**Performed using test cases**
Nature	Often implicit, like proofreading	**Very explicit**
Test Stub/driver	None is required	**May required either or both**
Verification/validatio n	Involves verification process	**Involves validation process**
Active/passive	A form of Passive testing	**Active testing**
Manual/automated	Usually performed manually	**Performed mostly using automation tools**
Main goal	Seeks to find defects in software	**Aimed at finding software failures**
Cost	Low cost of finding and fixing defects	**High cost of finding and fixing failures**
Execution time	Can be performed before the compilation	**Begin before completion of the software, usually on the smallest executable code section of a software**
Target component	Can be performed on software source code, design documents and models, functional and requirement pacifications, and any other documents	**Only performable on software source code.**

- **Testing levels.**

  Some researchers [19][20] have suggested the following levels of testing.

1. **Unit testing:** It falls under the category of White box, often known as component/module. The smallest testable component of a larger program confirms the functionality of a specific piece of code and satisfies the needs of the business.
2. **Integration testing:** It follows a top-to-bottom methodology, with each piece of the code being evaluated separately. Analysis of characteristics like functional, performance, and reliability requirements that are imposed on significant design elements is the primary duty.

3. **Functional testing:** It is also known as "Black Box." is done by providing trustworthy input, and findings are then evaluated. Functioning ensures that the system is fully functional by comparing expected output with actual output.
4. **Formation testing (White Box):** It focuses mostly on the internal architecture of the program, such as path coverage, statement coverage, control structure and program complexity, etc., enhancing both its appearance and usefulness.
5. **System testing:** It is regarded as a more specialized sort of testing that looks for flaws in the software components that are linked together. Additionally, it is a black box test that assesses the system's compliance with a predetermined criterion.
6. **Acceptance Testing:** It is a form of Black Box that the user performs to verify that a product, service, or system is ready and complies with business requirements as part of quality assessment processes. [1]. Because the user is unaware of the internal workings of the system, it is done after projects are finished by the developers before they pass them off to clients or users. The purpose is to provide the highest level of assurance for the operation of an error-free system with maximum reliability and efficiency [22].

The testing level evaluation is summarized in Table 5.

Table 5. Techniques for Testing Evaluation

Testing Level	Techniques	Tester	Scope	Time Consumption	Specification
Unit	White box	Developer	Classes	High	Low level
Integration	**White box, Black box, Grey box**	**Developer**	**Multiple classes**	**High, Average, Low**	**Low level and High level**
Functional	Black box	Independent	Entire product	Average	High level
Structural	White box	Developer	Entire product	High	Low level
System	Black box	Independent	Entire product	Average	Requirement Analysis
Acceptance	**Black box**	**User**	**Entire product**	**Average**	**Business Requirement**

Contrasting between the main testing levels shown in Table 6. [53]

Table 6. The Software Testing Levels compared

Criteria	Unit	Integration	System	Acceptance
Purpose	The correct working of the unit/module	The correct working of integrated units	**The whole system works well when**	Customer's expectations are met
Focus	Smallest testable part	Interface and interaction of modules	**Interaction and working of all**	Software working in accordance with given
Testing time	Once a new code is written	Once new components are added	**Once the software is complete**	Once the software is operationally ready
Performed by	Developer	Development team	**Testing team**	The development team and End-users
Testing technique	Usually Whitebox, and Greybox	Whitebox, and Blackbox	**Usually Blackbox, and Greybox**	Black-box testing
Automation	Automatable using JUnit, PHPUnit, TestNG	Automatable using Soap UI, Rest Client	**Automatable using Webdriver**	Automatable using Cucumber
Scaffolding	Complex (require drivers and/or stubs)	Moderate (may require drivers and/or stubs)	**No drivers/stubs**	No drivers/stubs are required

- **Procedure for manual testing.**

The process flow diagram for manual software/application checking is shown in Figure (2) The foundation of the manual is the software testing life cycle (STLC). STLC is followed by all software testing processes. To make the process easier to understand, it has been further broken down into 5 parts [21].

A. Analysis of the Bug or Flaw.

When a software tester receives a report of a bug or flaw, they analyze it and attempt to reproduce it in the software or application. A manual software tester estimates the number of hours needed to test a particular bug based on the analysis. A quality engineer reproduces the bug in many settings where it can be recreated during analysis, noting each one [22].

B. Writing cases for the defect in the unit test.

To help a developer understand where a bug might be replicated and provide a suitable repair, quality engineers prepare unit test cases for all the circumstances in which they believe a bug could be reproduced. A developer evaluates the software's fix according to the unit test cases provided by a quality engineer. The test cases that are written particularly for that flaw or problem are known as unit test cases [23].

C. Developer consultation and code deployment.

It is usually vital for quality engineers and developers to communicate often. Each other's task planning is aided by this. Before dumping the code into the software or application in this step, a quality engineer aids the developer in doing unit testing. The code is deployed (dumped) inside the software/application/build after unit testing by the developer and is then available for checking by the quality engineer [24].

D. White Box testing to test for the defect.

This stage entails evaluating every possible situation that the bug could be replicated and determining whether the developer's fix is effective across the board. Around the impact area, common sense is put to the test. To ensure that the program or application is sane, the basic functioning of the page or area where the repair was applied must be tested. Test the forms, modules, and other areas the developer identified as having a regression influence on that bug [25].

E. Reopen or close the defect

After checking, if a tester discovers no problems or discovers nothing else is broken as a result of the remedy applied, they close the bug and provide a report stating that everything is operational and the problem has been fixed. The problem is reopened by the tester with a report identifying what is broken and is given back to the developer to correct if, on the other hand, the remedy does not function in some circumstances or something is broken as a result of the fix. If a tester is testing a bug and discovers another bug that is unrelated to that bug, they report it as a new bug and the same processes apply to that newly reported bug [26].

4. Automated testing.

Programming testing is the most widely recognized approach to making a program in any programming or coordinating language that copies the standard examination stages with the help of an external motorization help gadget. Apparatus reserve ought to be made to test the source code that has proactively been used. It endeavors to help the testing process computerization. The creation of the program and the availability of test scripts are both seen as headway tasks; one interfaces with the real application, and the second to the items that will be used to test the application. These tests can be modernized [20].

- Challenging aspects of the system include running in the background, file logging, and database entry.
- Features that are frequently used but have a significant chance of error include payment systems, registrations, and others. Automation ensures quick faults because critical functioning tests typically take a few minutes.
- load tests, which check a system's performance under a heavy load of requests.
- Data requests, filling out multi-field forms, and verifying their preservation are a few instances of template actions.
- According to validation messages, incorrect data should be placed into the forms, and the validation should be evaluated.
- long-lasting situations from beginning to end.
- Confirmation of data includes accurate math calculations used in analytical or accounting procedures.
- examining the data search's accuracy.

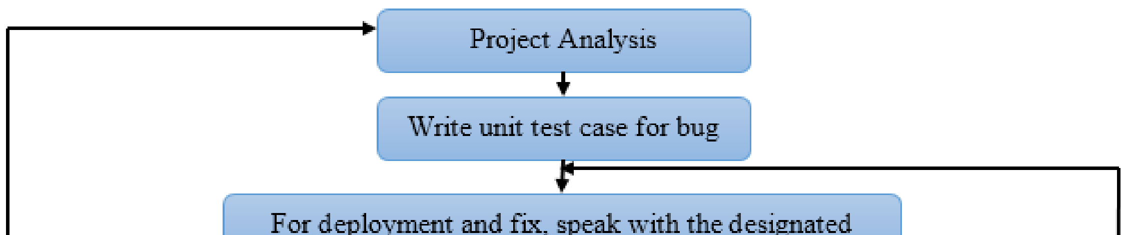

Project Analysis
Write unit test case for bug
For deployment and fix, speak with the designated

Figure 2. Manually executing tests

- **Automation Testing Tools.**

Software checking tools don't have any set standards. We must carefully select the testing automation tools that we use. Functional, loading, and administrative testing tools for both open source and commercial software are subcategories of software testing tools. Automated software testing solutions can be contrasted based on features like supported programming languages, operating systems, license types, supported browsers, prices, and supported programming languages, among others. By effectively using software testing tools, security, performance, correctness, and reliability testing may be carried out. The software and technology stack that will be employed, the specific testing requirements, the skill sets already present in the company, and the license cost of the tool all play a role in the tool selection process [27].

Different types of tools are employed for particular tasks. These are tailored to carry out the task assigned to them. A few necessary toolkits for every automated testing process include unit testing tools, functional testing tools, code coverage tools, test management tools, and performance testing tools. Unit testing tools like PHP Unit, N Unit, JMockit, and JUnit are examples. Selenium, Test Studio, HP Quick Test Professional, Tricentis Tosca Test suite, Test Complete, Waiter, and other functional testing tools are examples. Among the Code Coverage Tools are Clover, PITest, Code Cover, Atlassian, and Cobertura. Test Link, Test Environment Toolkit, Test Manager, and TET ware are a few examples of test management tools. The performance standard of the automated process is determined through performance testing tools such as HP Load Runner, JMeter, Silk Performer, and Rational Performance Tester [28].

A portion of the techniques utilized in robotized programming testing incorporates the ones recorded beneath: devices for dissecting experiments, inclusion analyzers to check the inclusion of testing, experiment generators that utilization item and information models with prerequisites, sensible and intricacy analyzers, code control framework instruments for social event data about the program while execution, imperfection following devices for following recognized mistakes and their status in being fixed, and apparatuses.

- **Contrasting between Manual and Automated testing.**

Test code inclusion and shortcoming identification rates in manual testing are not affected by code perceivabiy. Then again, with mechanized testing utilizing devices, restricted code deceivability regularly brings about low code inclusion and a low shortcoming recognition rate. Robotized testing is more costly than manual testing, especially right off the bat in the mechanization cycle. Hardware/programming testing instruments are both very costly. The profit from speculation will ultimately be productive. Robotized testing has no expense as opposed to manual testing, which demonstrates an expense each time a test is run in view of the length of the testing system [29].

The sum of the expenditures for both manual and automation testing is the overall cost of testing:

$CT = CM + CA + a$, where 'a' denotes other costs.

Testing software can't be fully automated. By cutting down on the time needed to create and run test cases, automation of the testing process can save testing costs. Table (7) summarized the contrast between Manual and Automated testing [30].

Table 7. Contrasting between Manual and Automated testing [30]

Manual Testing	**Automated Testing**
A piece of software is put through use case execution by a human tester.	**use a variety of tools to carry out use cases**
Allows for exploratory and random testing but is time-consuming	**Allows for exploratory and random testing but is time-consuming**
Relatively smaller investment	**Bigger invest men**
Prone to errors because of human intervention	**Highly robust and reliable**
Easily adaptable to changes	**When UI changes are made, scripts need to be updated.**
the necessity of human resources	**Requirement for testing tools and automation engineers**
Cost-effective for a smaller volume of testing	**Cost-effective for large volume testing**
Does not offer feasibility for performance testing	**Allows load testing, stress testing and other performance tests**
Suitable for AdHoc testing, exploratory testing, and situations where past tests are frequently changed	**Regression testing, load testing, and highly repeatable functional test scenarios are all appropriate.**

- **Swarm Intelligence (SI) and Machine Learning (ML) for Automated Testing.**

  A variety of SI-based algorithms are introduced in this topic, with an emphasis on their notable variants, benefits and drawbacks, and applications. Some of the algorithms in this category (CSA) are Artificial Bee Colony (ABC), Genetic Algorithms (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Differential Evolution (DE), Glowworm Swarm Optimization (GSO), and Cuckoo Search Algorithm [29].

A. Genetic Algorithm.

An inquiry streamlining procedure in view of the cycles of the regular choice interaction is known as the Hereditary Calculation (GA), which was first evolved by John Holland in 1975[31]. This calculation's key thought is to copy the possibility of "natural selection"; it mimics the cycles found in normal frameworks, where the solid will generally advance and get by while the frail will more often than not die. GA is a populace based procedure in which changes are assembled by how well their answers fit the issue. In order to focus their study on more exciting sections of the research universe, GA develops a starting set of potential solutions and recombines them.

B. Ant Colony Optimization.

A metaheuristic method called Ant Colony Optimization (ACO) was developed as a result of Marco Dorigo's 1992 Ph.D. thesis proposal of the Ant System (AS) [32][33]. In ACO, the amount of pheromones left behind by the ants as they go across the search area reveals how intense the trail is. Based on the course indicated by the intense trail, the ants decide where to go. One may think of the trail's intensity as the system's overall memory. Daemon actions are used to acquire comprehensive data that is impossible for a single ant to collect and utilize that data to decide whether additional pheromones need to be added to aid with convergence. The method is made resilient and flexible in a dynamic context using the decentralized control. Due to the flexibility that such a system offers in the event that an ant is lost or fails, having a decentralized system in ACO is crucial. These fundamental elements work in concert to produce shortest pathways through cooperative interaction [34].

C. Particle Swarm Optimization.

Kennedy and Eberhard first established the optimization method known as Particle Swarm Optimization (PSO) in 1995 [35]. It uses a straightforward mechanism to direct the particles as they look for worldwide best solutions, simulating swarm behavior in fish schooling and birds flocking. The three straightforward behaviors of separation, alignment, and cohesion were used by Del Valle and his co-authors [36][37] to characterize PSO. Alignment is the behavior of moving in the general direction of local flock mates, whereas separation is the behavior of avoiding the congested local flock mates.

The activity of moving toward the typical posture of one's immediate flock mates is known as cohesion. The population is initialized initially by the PSO method. The second stage involves determining each particle's fitness values, updating individual and global bests, and then changing the particles' velocity and position. Up until the terminating condition is met, steps two through four are repeated [37][38].

D. Differential Evolution.

This algorithm depends on the crossover operation, therefore it makes use of the mutation as a search mechanism and the selection operation to focus the search on promising areas in the search space. DE employs three characteristics—Target Vector, Mutant Vector, and Trail Vector—to iteratively create a new population. The vector containing the answer to the search problem is known as the target vector, while its mutation is known as the mutant vector and its offspring, the trail vector, is the result of the crossover operation between the target vector and the mutants vector.

E. Artificial Bee Colony.

This technique characterizes and separates fake specialists into three kinds: utilized honey bees, onlooker honey bees, and scout honey bees. To complete the calculation's methodology, different assignments are given to every one of these honey bees. The employed honey bees focus on a food source and recollect where that food supply is found. Since every working drone is connected to one and only one food source, the extent of working honey bees approaches the number of food sources. The working drone in the hive educates the observer honey bee regarding the area of the food supply. The nectar is then gotten from one of the food sources. The utilized honey bee is responsible for finding new nectar and food sources [39] [40].

F. Glowworm Swarm Optimization.

Glowing worm mobilization optimization (GSO), a novel method based on the International System of Functionalities that was introduced by Krishnanad and Goose in 2005 [41], aims to enhance multimodal functions. GSO uses what are known as glowworms as physical beings (agents).

By taking into account the following changes, GSO can be made better overall. 1) Including all agents in the neighborhood range expansion. All agents can travel in the direction of the agent with the best solution once the best solution has been found. As more agents are in the optimal solution range as a result of this step, exploitation efficiency may increase. 2) As few neighbors as feasible must be taken into account when calculating the neighborhood range in order to increase the GSO's convergence rate. Since there are fewer calculations needed to estimate the likelihood and direction of the GSO's movement, this action may shorten the GSO's processing time [42].

G. Cuckoo Search Algorithm.

Yang and Deb introduced the Cuckoo Search Algorithm (CSA) as one of the newest metaheuristic techniques in 2009 [43]. The behavior of cuckoo species, such as brood parasites, and Lévy flying characteristics, like those of some birds and fruit flies, served as the basis for this algorithm. Three fundamental guidelines or procedures are used in the implementation of CSA. First off, each cuckoo is only permitted to lay one egg per iteration, and the nest that the egg will be laid in is picked at random by the cuckoo. Second, the best nests and eggs are passed on to the following generation. Third, the number of available host nests is fixed, and a host bird uses probability pa [0, 1] to determine whether a cuckoo egg has been placed. In other words, the host has a choice as to whether to discard the egg or give up on the nest and start over. The final presumption can be roughly expressed as a percentage, pa, of the total n nests that are replaced by new nests using a new random solution. The algorithm can also be developed to a more complex level where each nest contains a number of eggs [44]. The specifics of the actions conducted in CSA are detailed based on these three main principles.

- **Use of Machine Learning.**

As per surveys, the utilization of AI will essentially affect the field of computerization testing. Our frameworks will want to evaluate current realities and information provided to them, make and run test situations on the said information, and afterward gain from the aftereffects of the experiments assuming we use AI in testing to empower them to do as such. The whole interaction would eventually further develop the testing cycle. The framework can make and convey more exact discoveries quicker than expected with the utilization of AI. In this particular sort of regulated AI, the framework is taken care of a bunch of information as preparing models. This information is incorporated by the framework, which utilizes it to figure or classes new information.

Six standard credits — physical work, test execution, mistake error, test scope, time included, and required programming abilities — are utilized to look at the customary and ML test computerization (Table 8).

Table 8. A contrast between traditional and ML-based automation [34]

Criterion	Conventional Test Automation	ML-Based Test Automation
Manual labor	increased use of manual labor	**Less manual labor is required**

Test performances	Needs to be changed by changing the effort	**demonstrates the ability to heal itself**
Error and inaccuracy	Because so much human labor is required, traditional test automation is prone to many mistakes and erroneous results.	**As ML-based automation replaces the majority of human effort, it is less likely to make mistakes or have inaccurate results.**
Test scope	The software's requirements or features need to be actively tested.	**Unusual features of the software being tested automatically might be examined.**
Time invested	In traditional automation, even minor UI changes necessitate script adjustments.	**There is no need to spend a lot of time scripting because ML adapts scripts to the variety of applications. required programming abilities**
Prerequisite programming skills	Traditional test automation requires the necessary technological expertise.	**Most ML-based tools can be utilized with only little technical expertise.**

- **Impact of Artificial Intelligence (AI) on Software Testing**

The software checking life cycle is a good way to identify the areas where AI approaches have shown to be helpful in software testing research and practice (STLC). The STLC phases from planning to reporting have all been significantly impacted by AI approaches. It is necessary to identify testing tasks for which extensive and important research has been conducted in order to examine the influence of AI on software testing (10) [54]. The benefits of AI techniques use in the field of software test automation are reported and grouped into larger categories and are shown in Table (9) together with their short description **[**55**]**

Table 9. Benefits of using AI techniques with software testing [23]

Benefit	Description
Manual effort reduction	While some of the testing procedures are fully automated, others are just semi-auton and involve more human interaction. By reducing the time and money needed for t creation, execution, and maintenance, the use of AI techniques replaces manual lab
Improved code coverage	The capacity to fully or substantially cover the assertions, branches, and transitio might be considered one of the reported benefits of the coverage. The publicatio compared the increased coverage to the methods that were previously in place.
Improved fault and vulnerability detection effectiveness	In comparison to current methods, generated test cases or oracles are, generally speaking, more effective and efficient in finding software faults.
Reusability of created test cases and test oracles	In the context of the publication, the reusability of test cases and oracles generated r be interpreted as being independent of one or more conditions, such as a particular (library, application, operating system, source code, or system model.
Test breakage repair	The ability of breakage repair capabilities to significantly reduce breakages and outperform existing solutions was reported in papers.
Avoiding duplicate activities while running the test	The use of AI approaches helps to reduce the number of unwanted activity transitio and restarts of the system under test, which increases test execution speed and accu
Improvement of existing solutions	A portion of these are improvements that a man-made intelligence based approach make to current experiment age, determination, and information age procedures produced text inputs are subject to the setting of the framework being tried and are created haphazardly; combinatorial blast is stayed away from during age; and experiments are picked in light of numerous targets as opposed to only one.
better assess the suitability of the produced test cases	In contrasting to other non-AI approaches, generated test cases are better able to ach the necessary test adequacy. The states covered, the viability, and the lack of duplication of the generated test ca are the criteria for sufficiency.

- **Software Reliability**

Accurate testing methods produce reliable software, and a thorough examination of testing methods will have a significant impact on the reliability and tolerability of the program. White box procedures give superior results for the reliability of the program, according to a contrast of the three testing methodologies—White Box, Grey Box and Black Box (Table 10). [56]

Table 10. A contrast of reliability of the three testing techniques

Software Testing Techniques	Defects Detection	Effect on the overall failure	Reliability
White Box	**73%**	**22%**	**78%**
Grey Box	63%	32%	56%
Black Box	42%	39%	32%

5. Conclusion

Software testing is obviously crucial to the software development process. Software testing is a process of continuous improvement of software, applications, and products. Testing is a condition for the final delivery of the software/application/product. Testing professionals can increase software quality by successfully conducting software testing with the help of their familiarity with these dynamic software testing methodologies and how they are used in software development. The research unequivocally shows that producing and deploying projects will be faster and more efficient with AI because it will enable tests to be built more quickly and problems found earlier. AI/ML will be able to calculate the probability of a build failure if and when the application code is modified. AI will improve and better predict troublesome areas that need more attention from testers by learning from our processes and tests. To sum up, creating and implementing AI-powered test automation is the best course of action for everyone.

6. Acknowledgments

The authors would like to thank the University of Mosul / College of Computer Science and Mathematics for their facilities, which have helped to enhance the quality of this work.

7. References

[1] M. M. Syaikhuddin, C. Anam, A. R. Rinaldi, and M. E. B. Conoras, "Conventional Software Testing Using White Box Method," *Kinet. Game Technol. Inf. Syst. Comput. Network, Comput. Electron. Control*, vol. 3, no. 1, pp. 65–72, 2018, doi: 10.22219/kinetik.v3i1.231.

[2] A. Verma, A. Khatana, and S. Chaudhary, "A Comparative Study of Black Box Testing and White Box Testing," *Int. J. Comput. Sci. Eng.*, vol. 5, no. 12, pp. 301–304, 2017, doi: 10.26438/ijcse/v5i12.301304.

[3] J. Bendickson, E. Liguori, and C. Midgett, "Compiler Design: Theory, Tools, and Examples," *Rowan Univ. bergmann@rowan.edu*, vol. 27, 2017.

[4] S. Nidhra and J. Dondeti, "BLACK BOX AND WHITE BOX TESTING TECHNIQUES - A LITREATURE REVIEW," *Int. J. Embed. Syst. Appl. Vol.2, No.2*, vol. 2, no. 2, pp. 29–50, 2012.

[5] D. Honfi and Z. Micskei, "Automated isolation for white-box test generation," *Inf. Softw. Technol.*, vol. 125, no. February, 2020, doi: 10.1016/j.infsof.2020.106319.

[6] I. Jovanovic, "Software Testing Methods and Techniques," *IPSI BgD Trans. Internet Res.*, vol. 5, no. 1, pp. 30–41, 2009, [Online]. Available: http://www.internetjournals.net/journals/tir/2009/January/Full Journal.pdf#page=31

[7] K. Mohd. Ehmer and K. Farmeena, "A Comparative Study of White Box , Black Box and Grey Box Testing Techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 6, pp. 12–15, 2012.

[8] M. Kaur and R. Singh, "A Review of Software Testing Techniques," *Int. J. Electron. Electr. Eng.*, vol. 7, no. 5, pp. 463–474, 2014, [Online]. Available: http://www.irphouse.com

[9] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," *Proc. -*

6th Int. Conf. Inf. Commun. Technol. Muslim World, ICT4M 2016, no. November 2017, pp. 177–182, 2017, doi: 10.1109/ICT4M.2016.40.

[10] C. Djaoui, E. Kerkouche, K. Khalfaoui, and A. Chaoui, “A graph transformation approach to generate analysable maude specifications from UML interaction overview diagrams,” *Proc. - 2018 IEEE 19th Int. Conf. Inf. Reuse Integr. Data Sci. IRI 2018*, pp. 511–517, 2018, doi: 10.1109/IRI.2018.00081.

[11] M. Albarka Umar, “Comprehensive study of software testing: Categories, levels, techniques, and types Software Testing View project System Analysis View project Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types,” *ResearchGate*, pp. 1–15, 2020, [Online]. Available: https://www.researchgate.net/publication/342538504

[12] S. Jat and P. Sharma, “Analysis of Different Software Testing Techniques,” *Int. J. Sci. Res. Res. Pap. Comput. Sci. Eng.*, vol. 5, no. 2, pp. 77–80, 2017, [Online]. Available: www.isroset.org

[13] Meenu and Y. Kumar, “Comparative Study of Automated Testing Tools: Selenium , SoapUI, HP Unified Functional Testing and Test Complete,” *J. Emerg. Technol. Innov. Res.*, vol. 2, no. 9, pp. 42–48, 2015, [Online]. Available: www.jetir.org

[14] F. Okezie, I. Odun-Ayo, and S. Bogle, “A Critical Analysis of Software Testing Tools,” *J. Phys. Conf. Ser.*, vol. 1378, no. 4, 2019, doi: 10.1088/1742-6596/1378/4/042030.

[15] H. Anjum *et al.*, “A Comparative Analysis of Quality Assurance of Mobile Applications using Automated Testing Tools,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 7, pp. 249–255, 2017, doi: 10.14569/ijacsa.2017.080733.

[16] T. Wala and A. Kumar Sharma, “International Journal of Computer Science and Mobile Computing Improvised Software Testing Tool,” *Int. J. Comput. Sci. Mob. Comput.*, vol. 3, no. 9, pp. 573–581, 2014, [Online]. Available: http://www.w3.org/2001/12/soap-envelope

[17] T. T and M. Prasanna, “Research and Development on Software Testing Techniques and Tools,” *Int. J. Curr. Eng. Technol.*, vol. 4, no. 4, pp. 1479–1493, 2018, doi: 10.4018/978-1-5225-7598-6.ch109.

[18] S. Gojare, R. Joshi, and D. Gaigaware, “Analysis and design of selenium webdriver automation testing framework,” *Procedia Comput. Sci.*, vol. 50, pp. 341–346, 2015, doi: 10.1016/j.procs.2015.04.038.

[19] K. Latha, “An Evaluation and Comparative Analysis of Software Testing Techniques and Tools,” *Int. J. Innov. Res. Sci. Eng. Technol.*, vol. 9, no. 2, pp. 185–190, 2020.

[20] I. Shuaibu, M. Musa, and M. Ibrahim, “Investigation onto the Software Testing Techniques and Tools: An Evaluation and Comparative Analysis,” *Int. J. Comput. Appl.*, vol. 177, no. 23, pp. 24–30, 2019, doi: 10.5120/ijca2019919685.

[21] V. Garousi and M. V. Mäntylä, “A systematic literature review of literature reviews in software testing,” *Inf. Softw. Technol.*, vol. 80, pp. 195–216, 2016, doi: 10.1016/j.infsof.2016.09.002.

[22] M. Albarka Umar, “Comprehensive study of software testing: Categories, levels, techniques, and types Software Testing View project System Analysis View project Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types,” *Int. J. Adv. Res. Ideas Innov. Technol.*, vol. 5, no. 6, pp. 32–40, 2020, [Online]. Available: https://www.researchgate.net/publication/342538504

[23] A. Trudova, M. Dolezel, and A. Buchalcevova, “Artificial intelligence in software test automation: A systematic literature review,” *ENASE 2020 - Proc. 15th Int. Conf. Eval. Nov. Approaches to Softw. Eng.*, no. Enase, pp. 181–192, 2020, doi: 10.5220/0009417801810192.

[24] Z. Khaliq, S. U. Farooq, and D. A. Khan, “Artificial Intelligence in Software Testing : Impact, Problems, Challenges and Prospect,” 2022, [Online]. Available: http://arxiv.org/abs/2201.05371

[25] T. Sheakh, “International Journal of Allied Practice , Research and A Comparative Study of Software Testing

Techniques Viz . White Box Testing Black Box Testing and Grey Box Testing," *Int. J. Allied Pract. Res. Rev. Website*, no. May, 2015.

[26] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review," *Softw. Test. Verif. Reliab.*, vol. 27, no. 8, pp. 1–20, 2017, doi: 10.1002/stvr.1639.

[27] D. S. N, S. D. S, D. Vijayasree, N. S. Roopa, and A. Arun, "A Review on the Process of Automated Software Testing," no. September, 2022, doi: 10.48550/arXiv.2209.03069.

[28] M. A. Umar and C. Zhanfang, "A Study of Automated Software Testing: Automation Tools and Frameworks," *Int. J. Comput. Sci. Eng.*, vol. 8, no. 06, pp. 217–225, 2019, doi: 10.5281/zenodo.3924795.

[29] P. K. Bhatia, "Impact of Software Testing Metrics on Software Measurement," *Int. J. Comput. Eng. Technol.*, vol. 8, no. 4, pp. 108–126, 2017,

[30] D. S. N;, S. D. S;, D. Vijayasree;, N. S. Roopa;, and A. Arun;, "A Review on the Process of Automated Software Testing," 2022.

[31] J. H. Holland, "Genetic Algorithms - Computer programs that 'evolve' in ways that resemble natural selection can solve complex problems even their creators do not fully understand," *Scientific American*. pp. 66–72, 1992.

[32] Y. R. Kaesmetan and M. V. Overbeek, "Ant Colony Optimization for Traveling Tourism Problem on Timor Island East Nusa Tenggara," *Indones. J. Artif. Intell. Data Min.*, vol. 3, no. 1, p. 28, 2020, doi: 10.24014/ijaidm.v3i1.9274.

[33] C. Gil, R. Baños, J. Ortega, A. L. Márquez, A. Fernández, and M. G. Montoya, "Ant colony optimization for water distribution network design: A comparative study," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6692 LNCS, no. PART 2, pp. 300–307, 2011, doi: 10.1007/978-3-642-21498-1_38.

[34] M. A. Salam, M. Abdel-Fattah, and A. A. Moemen, "A Survey on Software Testing Automation using Machine Learning Techniques," *Int. J. Comput. Appl.*, vol. 183, no. 51, pp. 12–19, 2022, doi: 10.5120/ijca2022921919.

[35] M. O. Okwu and L. K. Tartibu, "Particle Swarm Optimisation," *Stud. Comput. Intell.*, vol. 927, pp. 5–13, 2021, doi: 10.1007/978-3-030-61111-8_2.

[36] S. A. M. Al-bassam, "Learning the Neural Network using New Evolving Method to Solve Prediction Problems," *J. Kerbala Univ. , Vol. 8 No.4 Sci. . 2010*, vol. 8, no. 4, pp. 395–406, 2010.

[37] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Comput.*, vol. 22, no. 2, pp. 387–408, 2018, doi: 10.1007/s00500-016-2474-6.

[38] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comprehensive review of swarm optimization algorithms," *PLoS One*, vol. 10, no. 5, 2015, doi: 10.1371/journal.pone.0122827.

[39] Dervis KARABOGA, "AN IDEA BASED ON HONEY BEE SWARM FOR NUMERICAL OPTIMIZATION," *Tech. REPORT-TR06*, vol. 12 Suppl 1, no. 9, pp. 1–29, 2005

[40] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Glob. Optim.*, vol. 39, no. 3, pp. 459–471, 2007, doi: 10.1007/s10898-007-9149-x.

[41] Y. Zhou, J. Liu, and G. Zhao, "Leader glowworm swarm optimization algorithm for solving nonlinear equations systems," *Prz. Elektrotechniczny*, vol. 88, no. 1 B, pp. 101–106, 2012.

[42] Z. Li and X. Huang, "Glowworm Swarm Optimization and Its Application to Blind Signal Separation," *Math. Probl. Eng.*, vol. 2016, 2016, doi: 10.1155/2016/5481602.

[43] X. Yang, S. Deb, and A. C. B. Behaviour, "Cuckoo Search via L ´ evy Flights," *Ieee*, pp. 210–214, 2009.

[44] F. Zukhruf, R. B. Frazila, and W. Widhiarso, “A comparative study on swarm-based algorithms to solve the stochastic optimization problem in container terminal design,” *Int. J. Technol.*, vol. 11, no. 2, pp. 374–387, 2020, doi: 10.14716/ijtech.v11i2.2090.

تقنيات وأدوات اختبار البرمجيات: مراجعة

سكينة خليل عزت[1]، ندى نعمت سليم[2]

[1،2] قسم البرمجيات، كلية علوم الحاسوب والرياضيات، جامعة الموصل، الموصل، العراق

الخلاصة

ترتبط عملية تطوير البرمجيات ارتباطًا وثيقًا بعمليات التطوير. تكمن مشكلة تطوير البرنامج في أنه غالبًا ما يفتقر إلى الاختبار مما يؤدي إلى فشل البرامج. من أجل الحفاظ على منتج عالي الجودة يكون الاختبار أمرًا بالغ الأهمية. يمكن اختبار البرنامج باستخدام تقنيات اختبار الصندوق الابيض أو الصندوق الاسود أو الصندوق الرمادي في هذا التحقيق ، تمت مراجعة أنواع الاختبار. يستخدم إجراء الاختبار باستخدام اختبار الصندوق الابيض عددًا من منهجيات الاختبار بناءً على اختبار المسار ، بما في ذلك إنتاج المخططات الانسيابية وتقييم التعقيد الدوري واختبار المسار المستقل. نتيجة لذلك ، من الممكن تنفيذ تقنية اختبار مسار الأساس ونهج الصندوق الأبيض للاختبار. تضمنت هذه المراجعة عدة محاور ، وهي تعريف أدوات الاختبار، ثم تقنيات الاختبار بشكل عام ، وفوائد كل من هذه التقنيات، ومستويات الاختبار، وأخيراً خطوات إجراء الاختبار.