Volume: 12 Issue: 03 Pages: 4575-4584(2020) ISSN: 0975-0290

A Comparative Study of Dynamic Software Testing Techniques

Mubarak Albarka Umar
School of Computer Science and Technology, Changchun University of Science and Technology, Jilin, China
Email: 2018300037@mails.cust.edu.cn
Chen Zhanfang
School of Computer Science and Technology, Changchun University of Science and Technology, Jilin, China
Email: chenzhanfang@cust.edu.cn

ABSTRACT

The growing need for quality software makes software testing a crucial stage in Software Development Lifecycle. There are many techniques of testing software, however, the choice of a technique to test a given software remains a major problem. Although, it is impossible to find all errors in software, selecting the right testing technique can determine the success or failure of a software testing project. Knowing these software testing techniques and their classification is a vital key in selecting the right technique(s). Software testing can broadly be classified as static or dynamic, this paper presents a broad comparative study of the various dynamic software testing techniques. An explanation of the dynamic testing techniques, their advantages and disadvantages, as well as some of the commonly used types of testing under each technique are presented. Finally, a comparison of the dynamic testing techniques is also made to enable a clear and definite understanding of the techniques for the betterment of testing and subsequent improvement in software quality.

Keywords -**Software Testing, Software Testing Paradigm, Dynamic Testing Techniques, TestingTypes, Software Quality.**

Date of Submission: Nov 6, 2020 Date of Acceptance: Nov 23, 2020

1. INTRODUCTION

Software testing is defined as the process of evaluating a software program with the intent of finding fault or errors in software. Software testing is mainly performed to achieve the following three aims:

- To ensure that the software program can correctly perform its intended purpose [1].
- To verify that the software is fit for use [2].
- To achieve and maintain software quality as per a given standard [3], [4].

In Software Development Life Cycle (SDLC), software is not considered finished until it has passed its testing phase[5]. Software testing is very important for various reasons [6], [7]. The overall purpose of testing software is not to demonstrate that software is free of errors but to give confidence that the software is working well before installation[1]. There are various approaches of testing software that can be broadly classified as static or dynamic [2], [7]–[9]. The former involved methods that are used to determine software quality without reference to actual execution of the program while the later does the same thing but with a reference to actual program execution. This study focuses on the dynamic testing approach.
The dynamic testing comprises of experience-based testing techniques in addition to the three well-known techniques of testing software, namely, white box, black box, and grey box [7]. Precisely, our study is centered on all the dynamic testing techniques, their explanation, advantages and disadvantages, as well as an explanation of some of the commonly used types of testing under each of the dynamic testing approaches and lastly, a comparison of the four testing techniques. The study aims to provide a clear and definite explanation of the techniques for the betterment of testing and subsequent improvement in software quality. Some of the contributions of this work include:

- Thorough explanation of all the dynamic software testing techniques. Umar[17] is the closest work to cover most of the dynamic testing techniques from the available literature.
- A detailed literature review covering significant papers on software testing techniques, their common limitations are highlighted and comparison of the papers is also made.

The remaining part of this work is organized as follows; a review of the related studies from the literature along with their common limitation is presented in the next Section. Section III provides a brief explanation of the two software testing approaches, their comparisons, and the various activities and/or techniques under each. A broad overview of the software testing paradigm is also presented. Then Section IV provides a thorough explanation of the four dynamic testing techniques along with the common testing types under each, their pros and cons are also identified. A comparison of the testing techniques is made in Section V and finally, Section VI concludes the study.

2. LITERATURE REVIEW

Over the years, several works on software testing techniques were presented in the literature. In this section we review some of the significant works among them, specifically focusing on works that provide a detailed explanation of all or one of the three main software testing techniques in chronological order. Their common limitations are identified, a comparison of the reviewed literature is summarized in Table 1below.

Jovanovic [10] provides a short description of the two widely known testing techniques – white and black box techniques. Furthermore, the author explains broadly the various and frequently used forms of testing under each of the two testing techniques. Jovanovic's work is among the earliest works on software testing techniques and it provides the basis of many similar testing research that follow it, similarly. Khan and Khan [11] performed a study on the three most prevalent and commonly used software testing techniques for detecting errors. The three testing techniques are white-box testing, black-box testing, and grey-box testing. The authors provide a clear description of the techniques along with a brief explanation of some of the important testing types of each testing technique. They also compared the three testing techniques. Nidhra and Dondeti[12] conduct a study on black-box and white-box testing techniques with the aim of providing a clear explanation of the two different testing techniques, their advantages, and their usefulness. In conducting their studies, the authors carefully retrieved qualitative data, which mainly focus on testing techniques, their types, advantages, and case scenarios, from 29 articles and used them in succinctly explaining the testing techniques as well as the various testing types under those two testing techniques. Kaur and Singh [13] reviewed software testing techniques intending to analyze and compare many types of testing techniques to find out the best testing typefor finding errors from a software program. The authors concluded that the current testing technique knowledge is very limited and is based on impressions and perceptions. Jamil et al., [14] conducted a review on software testing techniques with the aims of discussing as well as improving the existing testing techniques for the better-quality assurance purposes. They briefly explain the three testing techniques, in addition to software release life cycles. The authors also proposed the use of a tool to enhance the existing testing processes. Sneha and Malle[15] researchedsoftware testing techniques and software automation testing tools intending to explain different testing techniques and types together with explaining some of the most used automation tools. Their research also highlighted the important role played by automation tools in comparison to the manual testing approach in software testing. Syaikhuddin et al., [16] conducted a study on conventional software testing with a specific focus on the white-box testing technique, they provide a simple explanation and case scenarios of some white box testing methods. The authors concluded that it will be necessary to find an experienced tester to conduct the testing process using White Box Testing method due to its complexity.Umar[17] presented a comprehensive study of software testing. The study provides a clear explanation and shows how the various testing categories, testing levels, the three testing techniques, and numerous types of testing relate to each other. The study provided the advantages and disadvantages of various testing methods and made some comparisons of different testing levels in addition to comparing the three testing techniques. The author also highlighted the role played by verification and validation in achieving software quality.

Table 1 - Literature Reviewed Summary

S/No.	Work 'Year	Dynamic/Static	Testing Levels	White box	Black box	Grey box	Experienced-based	Comparisons
1	[10] '09	×	×	✓	✓	×	×	×
2	[11] '12	×	×	✓	✓	✓	×	✓
3	[12] '12	×	✓	✓	✓	×	×	×
4	[13] '14	×	×	✓	✓	×	×	✓
5	[14] '16	×	×	✓	✓	✓	×	×
6	[15] '17	×	✓	✓	✓	✓	×	×
7	[16] '18	×	×	✓	×	×	×	×
8	[17] '19	✓	✓	✓	✓	✓	×	✓
9	*This work*	✓	×	✓	✓	✓	✓	✓

From the reviewed literature, there exists one common limitation in all the reviewed papers, it can be seen that none of the papers discussed the experienced-based testing technique. Majority of the existing literature mostly focused on white-box and black-box of dynamic testing techniques, none of them studied all the dynamic testing techniques, or provide a clear classification order of the various testing types under each technique with a detailed explanation of some of the most used testing types in each technique. This study aims to cover this gap.

3. SOFTWARE TESTING PARADIGM

There are various testing activities and techniques that are used in testing software to ensure it performs as expected. The entire testing activities and techniques can be classified as either static or dynamic testing [7], [9], they are both complementary to each other as they tend to find defects/failures effectively and efficiently. The static testing is the testing of software or its component at the specification or implementation level without executing the software, it is a powerful way of improving the quality and productivity of software development [7]. About 40% of software failures estimated to be due to static fault can be prevented using static testing [9]. On the contrary, dynamic testing involves the execution of the software or its component with a given set of test cases [7]. It's performed when the software is run and it may begin before the software is 100% complete such as testing a particular section of code. Most of the commonly known types of testing are under dynamic testing; and in most of the current literature, dynamic testing is referred to as "testing" while "verification activities" is the term used for static testing [2], [8]. Table 2 shows a comparison of the testing approaches, and Figure 1 provides a graphical view of the software testing paradigm.

Table 2 - Static VS Dynamic Testing [7], [9]

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/validation	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 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 specifications, and any other documents.	Only performable on software source code.

The dynamic testing involves any type of testing activity which requires running the software program or code [9]. White-box, black-box, and grey-box are three well-known forms of testing that require code execution, along with experienced-based testing they formed the four main dynamic testing techniques [7]. Any type of testing that requires code execution can be categorized in any of the aforementioned techniques. Each of the types of testing techniques determines the strategy that can be used in developing test cases for conducting the testing and in analyzing test results to achieve more effective testing[2]. They help identify test conditions that are otherwise difficult to recognize. There are several kinds of testing under each technique with each covering different aspects of software to reveal its quality. Utilizing all the testing techniques in testing a given software is not possible, but a tester can select and use more than one technique depending on the testing requirements, software type, budget, and time constraint. The higher the number of testing technique types combine, the better the testing result, coverage, and quality [10]. This study thoroughly discusses all the dynamic testing techniques, the various types of testing under each, their advantages and disadvantages, and a comparison of the techniques on many grounds is performed.

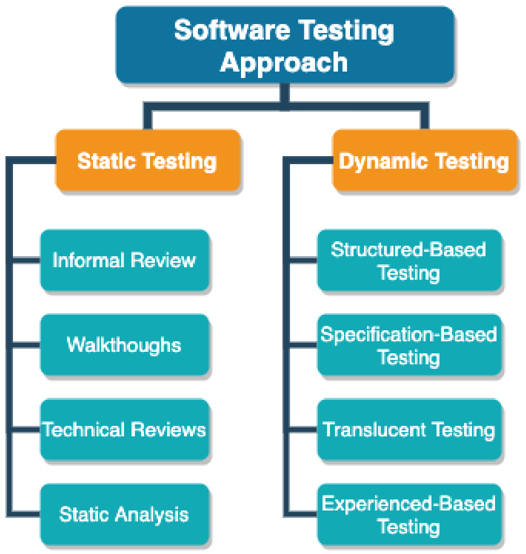


Figure 1 - Software Testing Paradigm [7], [17]

4. DYNAMIC TESTING TECHNIQUES

In software engineering, dynamic testing is a form of testing the dynamic behavior of code to examine the physical response from the software to variables that are not constant and change with time. In dynamic testing, the software must be compiled and run to check how it will perform in a run-time environment. It involves giving input values to the software and checking if the output is as expected by executing specific test cases which can be done manually or with the use of an automated process[18]. All the testing levels utilize dynamic testing and most of the testing activities in software testing are based on a dynamic testing approach. Specification-based, Structured-based, Translucent, and Experienced-based testing are the four major testing techniques [9], [17]. Table 3 shows the advantages and disadvantages of the dynamic testing approach.

Table 3 - Pros and Cons of Dynamic Testing

Advantages	Disadvantages
Can identify weak area in the runtime environment	Very time consuming because of the need to execute the software program often
Supports application analysis even if the tester does not have an actual code	Hard to find trained dynamic test professionals
Can verify the correctness of static testing as well as find vulnerabilities that are difficult to find with static testing.	Hard to track down the vulnerabilities in the code, and it takes longer to fix the problem. Therefore, fixing bugs becomes expensive

4.1. Structured-Based Testing

This is a dynamic testing technique in which internal structure and implementation of software under test are known to the tester. Structured-based testing required full knowledge of source code because test cases selection is grounded on the implementation of software entity, specifically the internal view of the system and tester's programming skills are used to design test cases [12]. Tester selects inputs to exercise program paths and compare the output with the expected output, the test oracle. Structured-based testing, although usually done at the unit level, is also performed at integration and system levels of the software testing process. Structured-based testing is also calledWhite Box, Transparent Box, Glass Box, Clear Box, Logic Driven, Open Box Testing.

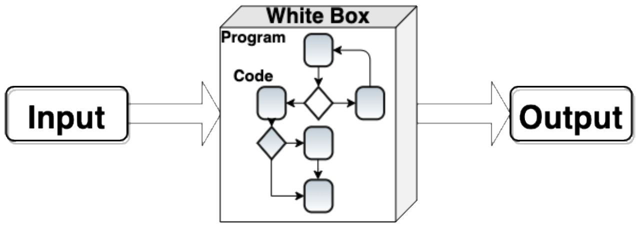


Figure 2 - Structured-Based Testing

Table 4 – Pros and Cons of Structure-Based Testing

Advantages	Disadvantages
Code optimization can be performed	Specialized tools are required such as debugging tools and code analyzers.
Easy to identify data and cover more test cases due to tester's knowledge of the code.	It's often expensive and difficult to maintain
Errors in hidden codes are revealed	Impossible to find and test all the hidden error and deal with them without going out of time

Structured-Based Testing Types

Some structured-based testing types include Control Flow, Data flow, Branch, Loop, Path Testing [19]. Some commonly used structured-based testing types are discussed below.

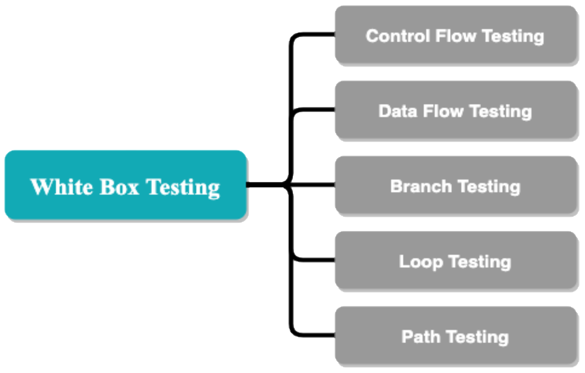


Figure 3 - Structured-Based Testing Types

4.1.1. Control-Flow Testing

Control flow testing is a type of structured-based testing that essentially checks the flow of control and order of executionof software using a control flow graph (CFG). It selects paths, nodes, and conditions from the CFG, and test cases are designed for executing these paths,with each path, node, or statements being traversed at least once during the testing process. Studying the control structure of the software is important in selecting and designing test cases [20]. Typically,a test case is anentire path from entry to exit nodes of the CFG. The selected set of paths are used to achieve a certain degree of testing thoroughness. Control-flow testing is most applicable at the unit level of software [21].

A typical CFG of a program comprises a set of nodes and edges, with each node representing a set of statements. There are five types of CFG nodes, viz.: unique entry and exit nodes, decision node (containing a conditional statement that can havea minimum of 2 control branches (such as a switch or if statements)), then merge node (which mostly represent a point where multiple control branches merge), and statement node having a sequence of statements. The control must flow from the first statement and exit from the last statement, and the CFG may have an additional edge between nodes for the reverse order flow of control (i.e. from the last to the first statement) [22]. There are several conventions for CFG models with subtle differences (e.g., hierarchical CFGs, concurrent CFGs). Control-flow testing supports the following test coverage criteria [22]:

- *Statement/Node Coverage*:Executes each statement in the program at least once
- *Edge Coverage*: Executes each statement in the program at least once using all possible outcomes at least once on every decision in the program.
- *Condition Coverage*: Executes each statement in the program at least once using all possible outcomes at least once on every condition in each decision.

Path Coverage: Executes each complete path in the program at least once. Except for loops, which usually has an infinite number of complete paths.

Table 5 – Pros and Cons of Control-Flow Testing

Advantages	Disadvantages
Catches 50% of all bugs caught during unit testing [21]	Cannot detect specification errors as well as Interface mismatches and mistakes
Very effective testing method for code that follows unstructured programming	Cannot catch all initialization mistakes
Enable experienced testers to bypass drawing CFG by doing path selection on the source	Time-consuming and required programming knowledge

4.1.2. Data-Flow Testing

Data-flow testing is a type ofstructured-based testing in which Control flow graph (CFG) paths are used to detect inappropriate definition or usage of data in predicates, computations, and termination (killing). It examines patterns in which a piece of data is used to identifies potential bugs [20]. Data flow testing searches for unreasonable things that can happen to data. Data flow anomalies are identified based on the associations between variables and values (unused initialized variables or uninitialized used variables). Data flow testing focuses on variables definition, use occurrence, and both predicate and computational use at different points within the program. There are two main data flow testingforms:(1) define/use testing which uses some simple rules and test coverage metrics, and (2) program slices that use segments of a program [23]. Data flow testing use the following test coverage criteria in creating test cases for the test [20]:

- All-defs (AD) coverage: which has a path from every definition to at least one use of that definition
- All-uses (AU) coverage: in which for every use of variable, there is at least one path from the definition to its use.
- All-c-uses (ACU) coverage: in which for every variable, there is a path from each of its definition to each of its c-use,any defined variable with no subsequent c-use is dropped from contention.
- All-c-uses/some-p-uses (ACU+P) coverage: in which for every variable, there is a path from each of its definition to each of its c-use. p-use is considered if there is any defined variable with no c-use following it.
- All-p-uses (APU) coverage: in which for every variable, there is a path from each of its definition to each of its p-use, any defined variable with no subsequent p-use is dropped from contention.
- All-p-uses/some-c-uses (APU+C) coverage: in which for every variable, there is a path from each of its definition to each of its p-use. c-use is considered if there is any defined variable with no p-use following it.

All-du-paths (ADUP) coverage: which covers all paths between definitions and uses for each def-use pair. It is thesuperset of all other data flow testing strategies and the strongest data-flow testing strategy. Moreover, this strategy requires the greatest number of paths for testing.

Table 6 – Pros and Cons of Data-Flow Testing

Advantage	Disadvantage
Can define intermediary Control flow analysis criteria between all-nodes and all-paths testing	Unscalable Data-Flow Analysis algorithm for large real-world programs
Handles variable definition and usage	Test case design difficulties compared with control flow testing.
It spans the gap between all paths and branch testing	Infeasible test objectives which might lead to wastage of time on testing in vain [24].
Identify multiple variable declarations	Can have an infinite number of paths due to loops

4.2. Specification-Based Testing

This is a dynamic testing technique in which the internal structure and implementation of software under test are not known to the tester, and it can be functional (such as integration testing) or non-functional (such as performance testing), though usually functional. It is called specification-based because test cases are built around requirement specifications and user stories. The specification-based testing emphasizes evaluating the fundamental aspects of software using thorough test cases, and generally, on maintaining the integrity of external information [19]. For a given test case, the tester verifies proper acceptance of inputs and correct production of outputs against its test oracle. Specification-based testingcan be applied at all levels of software testing processes such as Unit, Integration, System and Acceptance Testing levels, although done mostly on System testing and Integration testing.This testing is also called Opaque, Functional, Black-box, Close-box, Behavioral, and Input-Output testing.

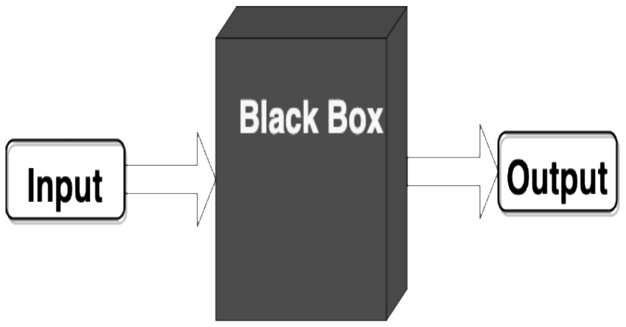


Figure 4 - Specification-Based Testing

Table 7 – Pros and Cons of Black-box Testing

Advantages	Disadvantages
Code knowledge is not required, tester's perception is very simple	Limited coverage, few test scenarios are designed/performed.
User's and developer's view are separate	Some parts of the backend are not tested at all.
Access to code is unrequired, quicker test case development	Inefficient testing due to the limited knowledge of code possesses by a tester.
Efficient and suitable for large parts of code	Test cases are difficult to design without clear specification

Specification-Based Testing Types

Some specification-based testing types include equivalence Partitioning, Cause-Effect Graph, Fuzzing, Boundary Value Analysis, Decision Table, State Transition, Orthogonal Array, and All Pair Testing [11]. Some common specification-based testing types are discussed below.

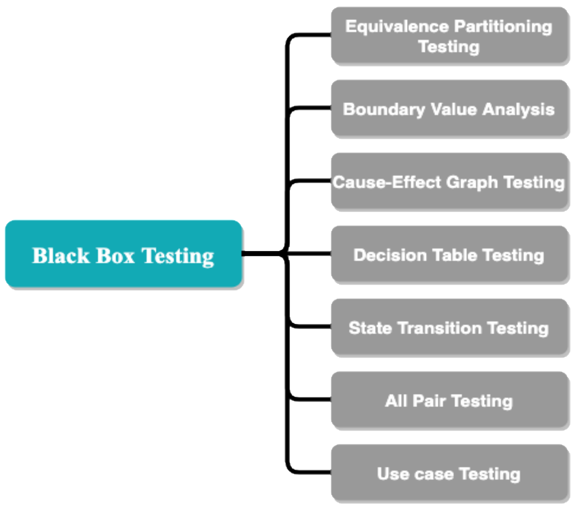


Figure 5 - Specification-Based Testing Types

4.2.1. Equivalence Partitioning Testing (EP)

This is a specification-based testing technique that dividesthe input domain into different equivalence classes to reduce the number of test cases andselects one element from each equivalence class (EC) as a test case. It is used to avoid test redundancy and give a sense of complete testingsince exhaustive testing is not possible. EC Testing can be either weak or strong. In Weak Equivalence Class Testing (WECT), some test cases are defined by choosing one variable value from each equivalence class and then taking the maximum value from the chosen variables, while test cases in Strong Equivalence Class Testing (SECT) are defined based on the cartesian product of partition class, i.e., testing all interactions of all equivalence classes [25].

Table 8 – Pros and Cons of Equivalence Partitioning Testing

Advantages	Disadvantages
Provide a sense of complete testing and eradicates the need for exhaustive testing	Suitable only for range-wise and discrete values input data
Enables large domain of inputs or outputs coverage with a smaller subset selected from an equivalence class	Assumes that the data in the same equivalence class is processed in the same way by the system
Avoid test redundancy by selecting a subset of test inputs from each class.	Can't handle boundary value errors. Need to be supplemented by boundary value analysis

4.2.2. Boundary Value Analysis Testing (BVA)

This is a specification-based testing technique that aims at finding software errors at the boundaries of equivalence classes. Unlike the Equivalence Partitioning technique that uses only the input domainin creating test cases, the BVA uses both input and output domains. The BVA complements EPtesting such that while EP tests program with test cases from within equivalence classes, the BVA focuses on testingthe program using test cases at and near the boundaries of equivalence classes [25]. Furthermore, test cases derived using either of the two techniques may overlap.

Table 9 – Pros and Cons of Boundary Value Analysis Testing

Advantages	Disadvantages
Complements Equivalence Partitioning testing by handling equivalence class boundary errors.	Generate a high number of test cases
Works well with variables that represent bounded physical quantities	Can't be used for Boolean and logical variables
Can be used at unit, integration, system and acceptance test levels	Function nature and variable meaning are not considered
Computationally less costly in creating test cases	Not that useful for strongly-typed languages

4.3. Translucent Testing

Translucent testing also known asGrey Box testing, is another dynamic testing technique that takes the straightforward technique approach of specification-based testing and combines it with the code-targeted oriented testing approach of structured-based testing. Some knowledge, usually of the part to be tested, of the internal working of the software is required in designing test cases at a specification-based level. More understanding of the internal implementation of software is required in grey-box testing than in specification-based testing, but less compared to structured-based testing [19]. Grey box testing is much more effective in integration testing and is the best approach for functional or domain testing, it is also a perfect fit for testing web-based applications [26].

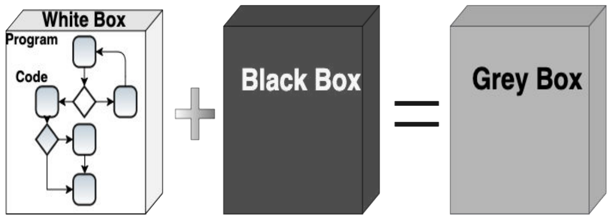


Figure 6 - Translucent Testing

Table 10 – Pros and Cons of Grey-box Testing

Advantages	Disadvantages
Provides combined benefits of both white-box and black-box testing	Complete white-box testing cannot be done due to inaccessible source code/binaries
Can handle design of complex test scenario more intelligently	Defect association is difficult in distributed systems.
Maintain boundary between independent testers and developers	Not suitable for algorithm testing.

Translucent Testing Types

Some grey-box testing types include Orthogonal Array, Regression, Pattern, and Matrix Testing. Some of these testing types are discussed.

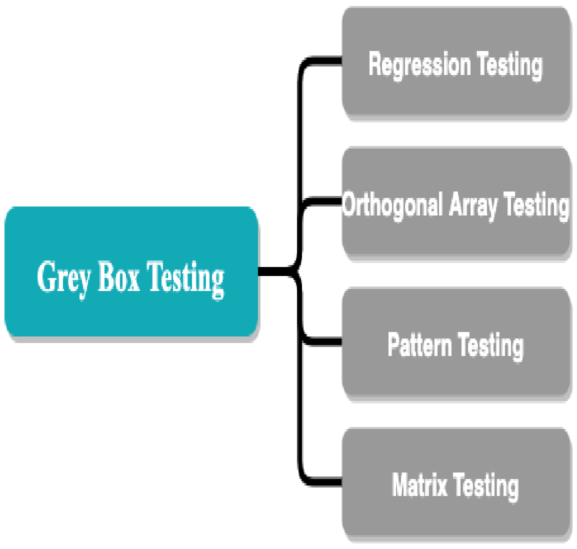


Figure 7 - Translucent Testing Types

4.3.1. Regression Testing

Regression testing is a grey-box testing strategy that is performed every time changes are made to the software to ensure that the changes behave as intended and that the unchanged part of the software is not negatively affected by the changes. Errors that occurred at unchanged parts of the software are called regression errors. Regression testing starts with a (possibly modified) specification, a modified program, and an old test plan (which requires updating) [27].

Table 11 – Pros and Cons of Regression Testing

Advantages	Disadvantages
Tests can be automated thereby saving time and improving the quality of software.	Tedious and time-consuming if done without automated tools
It ensures that a fix doesn't adversely affect working functionality.	Testing is required even on making slight changes to the program
Improves and maintain software quality	One of the main causes of software maintenance expensiveness.

4.3.2. Orthogonal Array Testing (OAT)

This is a grey-box testing type that usesindependent pair-wise combinations of data or entities as test input parameters to increase the testing scope. OAT is handy when maximum coverage is required with minimum test cases and a huge number of test data having many permutations and combinations. It’s extremely valuable for testing complex applications and e-comm products [28].

Table 12 – Pros and Cons of Orthogonal Array Testing (OAT)

Advantages	Disadvantages
Test pair-wise combinations of all the selected variables	Increase in Test case complexity as input data increases
Creates fewer Test cases which cover the testing of all the combination of all variables.	Tedious and time-consuming if done manually.
Improves productivity because of reduced test cycles and testing times.	

4.4. Experienced-Based Testing

This is atechnique of testing software with the help of experience gained through several years of working. The working experience here can be from testing or development experience, dealing with similar software or previous release of the same software, as well as experience in the respective domain. The knowledge of tester is very useful in designing the test cases. Experience-based testing is usually performed on software with fewer risks [29].

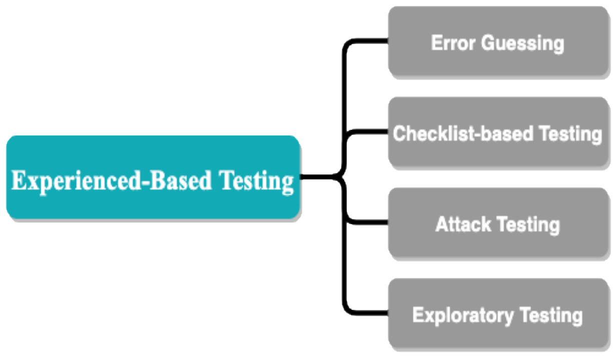
Table 13 - Pros and Cons of Experienced-Based Testing

Advantages	Disadvantages
Very crucial in the absence of requirements and specifications	Not ideal for high-risk software
Performable with limited knowledge of software	Required highly skilled and experienced testers
Doable when there is restricted time for testing	Software may still require a more formal and thorough testing

Experienced-Based Testing Types

Generally, there are four types of testing techniques where the experience of a tester plays a major role. These are Error Guessing, Checklist-based testing, Attack testing, and Exploratory testing. Some of these testing types are discussed.

Figure 8 - Experienced-Based Testing Types



4.4.1. Error Guessing

This is a testing technique in which an experienced tester applies his/her skills, gained knowledge, and experience to identify the vulnerable areas of the software that are likely to be affected by potential defects. Thereafter, tester marks each area according to its prone to defects with low-risk, medium-risk,or high-risk, and prepares the test cases to locate the possible existence of defects. This technique may be considered as a risk analysis method and it has no explicit testing rules; test cases can be designed depending on the situation, either drawing from functional documents or when an unexpected/undocumented error is found during testing operations [30].

Table 14 - Pros and Cons of Error Guessing

Advantages	Disadvantages
Very effective when used with other formal testing techniques	Person dependent and thus the experience of the tester controls the quality of test cases.
Uncovers defects which will otherwise be impossible to find using formal testing.	Not for newbie testers, only experienced testers can perform this testing.
Tester's experience saves a lot of time and effort.	Provides no guarantee to the achievable level of quality

4.4.2. Exploratory Testing

This a form of software testing that is concisely described as simultaneous learning, test design, and test execution. It is formal testing where a skilled tester uses his/her skill and experience to investigate bugs or errors in software without any preparation or a particular schedule. The tester identifies the proper working ofthe functionality of the software or otherwise, and consequently applies his/her skill and ability for exploring the application and set the test scenarios for the higher execution as well. The tester's knowledge of the software and that of testing approaches determines the quality and effectiveness of the testing result. It provides work flexibility and freedom to the tester. Exploratory testing is a progressive learning approach that helps in performing maximum testing with minimal planning. It is ideal when there are inadequate specifications or requirements and severely limited time [31].

Table 15 - Pros and Cons of Exploratory Testing

Advantages	Disadvantages
Require less preparation and/or no schedule	Testing cannot be reviewed
Critical defects are found very quickly and reveal bugs typically overlooked by other testing methods.	It is difficult to keep track of which tests have been run.
More intellectually stimulating than the execution of scripted tests at run time	Dependable on the skill and knowledge of the tester.

5. COMPARISON OF DYNAMIC SOFTWARE TESTING TECHNIQUES

It is important to note that there is no particular testing technique that is always better, however, depending on testing requirements, needs, budget, among others, one technique can have advantages over others. Furthermore, in testing any software, exploring and combining many testing techniques can help better in eliminating more bugs thereby increasing the overall quality of the software than sticking to one technique. Table 16below presents comparisons of the four discussed testing techniques based oncertain criteria.

Table 16 – Comparison of Testing Techniques

Criteria	**Structured-based**	**Specification-based**	**Translucent**	**Experienced-based**
Required knowledge	Full knowledge of internal working of software.	Knowledge of internal working of software is not required.	Limited knowledge of the internal workings of software.	Require thorough knowledge of testing and domain
Performed by	Usually testers and developers.	End-users, developers, and testers	End-users, developers, and testers	Experienced testers and end-users
Testing focus	Internal workings, coding structure, and flow of data and control.	Evaluating fundamental aspects of the software	High-level database diagrams and data flow diagrams.	Not very structured, depends on tester's instincts.
Granularity	High	Low	Medium	Low
Time consumption	Very exhaustive and high time-consuming	Exhaustive and low time-consuming.	Partly time-consuming and exhaustive.	Random and the least time-consuming
Data domain testing	Data domains and internal boundaries can be better tested.	Can be performed through trial-and-error method.	Can be done on identified Data domains and internal boundaries	Less emphasis is given to data domain compared to internal coding
Algorithm testing	Suitable for testing algorithms.	Unsuitable for testing algorithm.	Inappropriate for testing algorithms.	Less suitable for testing algorithm
Also known as	White-box, Transparent-box, Open-box, Logic-driven, or code-based testing.	Closed-box, data-driven, functional, or Black-box testing.	Grey-box testing	N/A

6. CONCLUSION

Delivering quality software is the main goal of any software project. Software Testing has been widely used and remains a truly effective means of assuring the quality of software. In this paper, the most important and most applied dynamic software testing techniques, their advantages and disadvantages are discussed, their comparisons are also presented. Learning about and effective usage of these dynamic software testing techniques in software development will help testers to carry out a successful software testing thereby improving software quality.

REFERENCES

[1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing 3rd Edition*, 3rd Ed. Canada: John Wiley & Sons, Inc., 2012.

[2] L. Luo, 'A Report on Software Testing Techniques', Pittsburgh, USA, 2001.

[3] E. Miller, *Software testing & validation techniques*. [Washington D.C.]: IEEE Computer Society Press, 1981.

[4] D. R. Graham, 'TESTING, VERIFICATION AND VALIDATION', *Int. J.*, vol. XVI, pp. 1069–1101, 1979.

[5] A. Dennis, B. H. Wixom, and R. M. Roth, *Systems Analysis and Design 5th Edition*, 5th Editio. USA: John Wiley & Sons, Inc., 2012.

[6] 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: https://doi.org/10.5281/zenodo.3924795

[7] D. Graham, E. Van Veenendaal, I. Evans, and R. Black, *Foundations of Software Testing: ISTQB Certification*. 2008.

[8] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[9] W. L. Oberkampf and C. J. Roy, *Verification and validation in scientific computing*. Cambridge University Press, 2010.

[10] I. Jovanovic, 'Software Testing Methods and Techniques', *IPSI BgD Trans. Internet Res.*, vol. 5, no. 1, pp. 30–41, 2009.

[11] M. E. Khan and F. Khan, '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.

[12] S. Nidhra and J. Dondeti, 'Black Box and White Box Testing Techniques', *Int. J. Embed. Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012.

[13] M. Kaur and R. Singh, 'A Review of Software Testing Techniques', *Int. J. Electron. Electr. Eng.*, vol. 7, no. 5, pp. 463–474, 2014.

[14] M. A. Jamil, M. Arif, N. Sham, A. Abubakar, and A. Ahmad, 'Software Testing Techniques : A Literature Review', in *2016 6th International Conference on Information and Communication Technology for The Muslim World Software*, 2016, pp. 177–182.

[15] K. Sneha and G. M. Malle, 'Research on software testing techniques and software automation testing tools', in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, 2017, pp. 77–81.

[16] M. M. Syaikhuddin, C. Anam, A. R. Rinaldi, and M. E. B. Conoras, 'Conventional Software Testing Using White Box Method', *Kinetik*, vol. 3, no. 1, p. 67, 2018.

[17] M. A. Umar, 'Comprehensive study of software testing : Categories , levels , techniques , and types', *Int. J. Adv. Res. Ideas Innov. Technol.*, vol. 5, no. 6, pp. 32–40, 2019. DOI: 10.36227/techrxiv.12578714

[18] G. J. Myers, *The art of software testing, Second Edition*, 2nd Ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.

[19] E. Khan, 'Different Forms of Software Testing Techniques for Finding Errors', *Int. J. Comput. Sci. Issues*, vol. 7, no. 3, pp. 11–16, 2010.

[20] J. Badlaney, R. Ghatol, and R. Jadhwani, 'An Introduction to Data-Flow Testing', *Control*, pp. 1–8, 2006.

[21] S. Mancoridis, 'CS576 Dependable Software Systems - Topics in Control-Flow Testing'. [Online]. Available: https://www.cs.drexel.edu/~spiros/teaching/CS576/slides/2.control-testing.pdf. [Accessed: 05-May-2019].

[22] N.-W. Lin, 'Software Testing (CS5812) - Control Flow Testing'. [Online]. Available: https://www.cs.ccu.edu.tw/~naiwei/cs5812/st4.pdf.

[23] M. New, 'Data Flow Testing Swansea University UK'.

[24] T. Su *et al.*, *A Survey on Data-Flow Testing*, vol. 50, no. 1. 2017.

[25] L. Briand, 'Software Verification and Validation - WBT', 2010. [Online]. Available: https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF4290/v10/undervisningsmateriale/INF4290-WBT.pdf. [Accessed: 03-May-2019].

[26] 'Software Testing Class - Grey box'. [Online]. Available: https://www.softwaretestingclass.com/gray-box-testing/.

[27] L. Briand, 'Software Verification and Validation (INF4290) - Regression Testing', 2010. [Online]. Available: https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF4290/v10/undervisningsmateriale/INF4290-RegTest.pdf.

[28] Alex Samurin, 'Explore the World of Gray Box Testing', 2003. [Online]. Available: http://extremesoftwaretesting.com/Articles/WorldofGrayBoxTesting.html. [Accessed: 19-May-2019].

[29] G. Bath and J. McKay, *The Software Test Engineer's Handbook : A Study Guide for the ISTQB Test Analyst and Technical Test Analyst Advanced Level Certificates*, 1st ed. Rocky Nook, 2008.

[30] B. Homès, *Fundamentals of Software Testing*. John Wiley and Sons, 2013.

[31] C. Kaner, 'A Tutorial in Exploratory Testing. Software Engineering', 2008.

Authors Biographies

MUBARAK ALBARKA UMAR was born in Dutsinma, Katsina State, Nigeria. He received the BSc. (Honors) Degree in Software Engineering from the University of East London in 2015, and MEng. Degree in Computer Applied Technology from the Changchun University of Science and Technology, China in 2020.
He is a staff of Katsina State Institute of Technology and Management (KSITM), Katsina, Nigeria. He previously did a one-year national service at Usmanu Danfodiyo University, Sokoto (UDUS), Nigeria from 2015 to 2016. He has published 7 articles in peer-review international journals. His research interests include data mining, soft computing, software engineering, network security and fuzzy logic.

ZHANFANG CHEN, Ph.D., Associate Professor of School of Computer Science and Technology in Changchun University of Science and Technology, China. His research interests include network engineering, software engineering, computer architecture, data mining, soft computing, and software engineering.
He is currently a teacher of Changchun University of Science and Technology and has worked for 15 years. He's worked on over 20 projects and published 20 articles in peer-review international journals.