Comprehensive Study of Software Testing: Categories, Levels, Techniques, and Types

Mubarak Albarka Umar
School of Computer Science and Technology,
Changchun University of Science and Technology,
7186 Weixing Road, Jilin, China
Email: 2018300037@mails.cust.edu.cn

ABSTRACT
Software Testing is the process of evaluating a software program to ensure that it performs its intended purpose. Software testing verifies the safety, reliability, and correct working of a software. The growing need for quality software makes software testing a crucial stage in Software Development Lifecycle. There are many methods of testing software, however, the choice of method to test a given software remains a major problem in software testing. Although, it is often impossible to find all errors in software, employing the right combination of methods will make software testing efficient and successful. Knowing these software testing methods is the key to making the right selection. This paper presents a comprehensive study of software testing methods. An explanation of Testing Categories is presented first, followed by Testing Levels and their comparison, then Testing Techniques and their comparison. For each of the Testing Levels and Testing Techniques, examples of some testing types and their pros and cons are given with a brief explanation of some of the important testing types. Furthermore, a clear and distinguishable explanation of two confused and contradictory terms (Verification and Validation) and how they relate to Software Quality is provided.

Keywords: *Software Testing, Testing Categories, Testing Levels, Testing Techniques, Testing Types, Software Quality, Verification and Validation.*

1. INTRODUCTION

Software testing is an integral phase in Software Development Life Cycle (SDLC) process [1], it involves many technical and non-technical aspects (such as specification, design, implementation, installation, maintenance and management issues) in software engineering [2]. Around 50% of software projects' development time and effort are put in software testing [3], [4], [2]. Software testing is defined as the process of evaluating a software program with the intent of finding fault or errors in software. Testing is done to; ensure that a software performs its intended purpose correctly [3], access, achieve and preserve the quality of a software [4], [5], and thereby verify that a software is fit for use [2]. In SDLC, the software is not considered finished until it has passed its testing [6], and the earlier an error is detected, the cheaper it is to fix it. The overall purpose of testing is not to demonstrate that software is free of errors but to give confidence that the software is working well before installation.

The "software we write [develop] today potentially touches millions of people" [3] across various walks of life and has become an integral part of our routines, this indicates the need for safe and reliable software. Unfortunately, humans are prone to err, and so the fundamental facts of humans' core involvement in software development make errors an inevitable inclusion in a software [4]. Software errors (bugs) can cause serious effects in live operation [4] and even death [7]. It is important to treat such errors early because they get costlier with progress in the development phase. For instance; a report released by the National Institute of Standards and Technology (NIST) estimated that software bugs are costing the USA economy $59.5 billion annually [8], Jones also highlighted in his survey [9] that $500 billion is lost annually due to poor software quality and the cost be reduced through testing the software. The eminent and massive effects of software bugs cannot be overestimated and hence, the need for software to be tested before delivered.

In the context of Software Quality, Verification and Validation (V&V) are often confusing terms. However, testing help in achieving quality software through Verification and Validation (V&V) methods. Verification is a *Quality Control* (QC) process that is concerned about building the software right, and Validation is a *Quality Assurance* (QA) process that is concerned with building the right software. Thus, Verification checks the conformity to the standard of software by verifying the correctness of one life cycle's deliverable transformation to the next while Validation checks back against the requirements of the customers. Verification is an internal process which involves set of activities to ensure that software correctly implements specific functions, it is usually done by the development team while, Validation requires some external process and involves set of activities to ensure that the developed software is traceable to customer requirements [10], it is mostly done with the stakeholders to provide a degree of software assurance. Verification usually begins before Validation and then they run in parallel until the release of the software. The use of V&V methods during software development helps in the early detection of an error, and hence, it can be fixed at a low cost [4].

There are two testing categories (approaches): Static and Dynamic [2], [11], [12]. There are generally three main software testing techniques which are all under dynamic testing approach [12]: White-box, Black-box and Grey-box testing [13], [14], [15]; each of the dynamic testing can be performed at different testing levels and they comprise of several types of testing. There are four general software testing levels: unit testing, integration testing, system testing, and acceptance testing [6], [2], [14],[16] and various types of testing comes under these levels [17].

The remaining part of this paper is organized as follows; Section Two presents the software testing approaches, followed by software testing levels, how they relate to the SDLC process, and their comparison in Section Three. Then in Section Four software testing techniques are thoroughly discussed, their comparison is also provided. Section Five provided a brief explanation of some of the most important types of testing and finally, the conclusion is made in Section Six.

2. TESTING CATEGORIES (APPROACHES)

Static and Dynamic testing are the two testing approaches that are occasionally inseparable but are mostly discussed separately [2]. The Static testing approach is done without executing the program and is called "verification activities", while the Dynamic testing approach involves executing the program with real inputs, most of the current literature refers to the dynamic testing as "testing" [11].

Static Testing Approach: involves source code only and it deals with program and symbolic analysis, model checking, error handling, and code inspection to ensure functional requirements, design, and coding standards are observed and estimate software quality without any reference to actual executions [2]. Desk checking, Code walkthrough, and Formal inspections are the commonly used techniques here [18], [19].

Dynamic Testing Approach: involves actual code executions [11] to ascertain and/or approximate software quality and it deals with a combination of inputs, use of structurally dictated testing procedures, and automation of testing environment generation [2] to test the internal design of the software. Most of the testing we perform is in this category as seen in *Figure 1*.

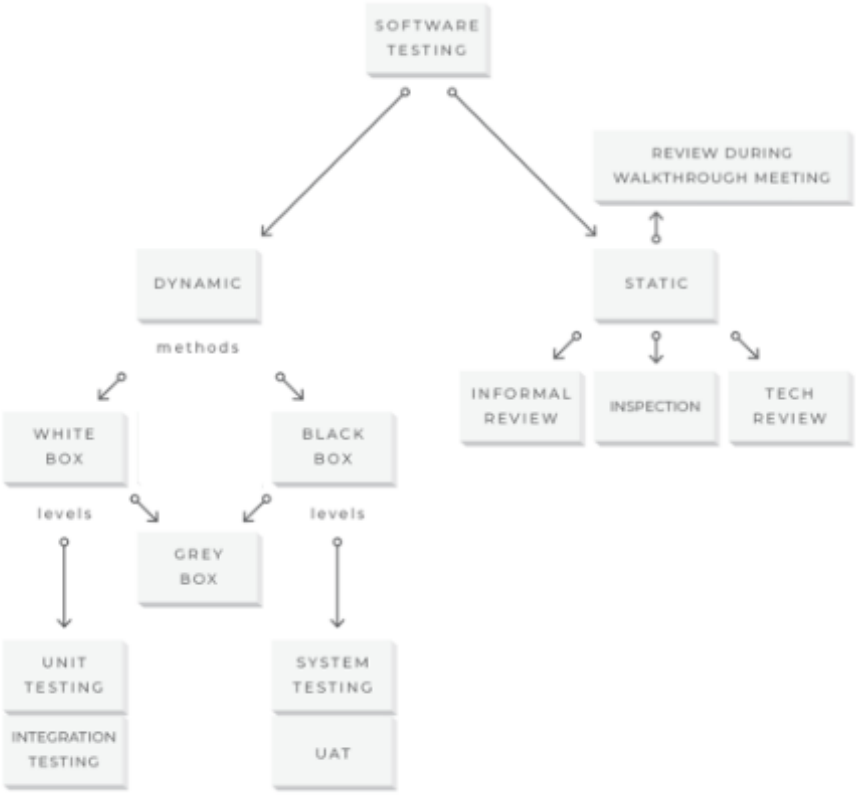

Figure 1: Software Testing Categories

3. SOFTWARE TESTING LEVELS

Unit testing: This testing emphasizes on individual units or modules in isolation. It is a testing in which the smallest testable portion of software is tested to verify its functionality against its specification. The unit can be a constructor or destructor at class level in an object-oriented environment [20] or a structure in procedural programming paradigm. Control-flow testing and data flow testing are some of the types of Unit testing. Unit testing is usually done by developers [6].

Integration Testing: involves testing two or more combined units that must work together to ensure an error-free flow of control and data (such as consistency of parameters, file format, and so on) among combined units and their overall correct design and integration. User interface, use-case, interaction, and big bang (integrate and test all modules at once) are some of the integration testing types. This kind of testing is performed by testers [6].
System Testing: involves testing an integrated complete software to check against its compliance with its requirements. It verifies the overall interaction of components to ensure the unanimous working of all modules and programs without error. It involves various types of both functional (tests functionality of software) testing and non-functional (tests quality of software) testing such as performance, reliability, usability, and security testing. System testing is performed by the testing team [6].
Acceptance Testing: This testing is performed to validates the software against customer requirements. This testing is done to ensure that the software does what the customer wants it to do and check the acceptability of the system. User Acceptance Testing (UAT), as sometimes called, comprises of two testing types: Alpha testing: is a testing performed by both development team and users using made-up data, and Beta testing in which users start using the software with real data and carefully observer the software for errors [6].

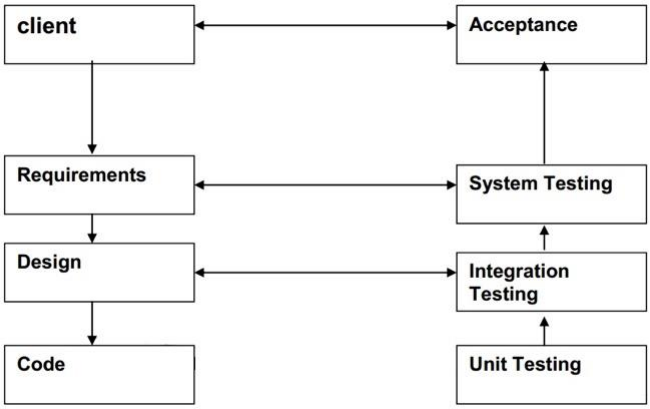

Figure 2: Software Testing Levels

Table 1: The Software Testing Levels compared [12].

Criteria	**Unit**	**Integration**	**System**	**Acceptance**
Purpose	Correct working of unit/module	Correct working of integrated units	Whole system works well when integrated	Customer's expectations are met
Focus	Smallest testable part	Interface and interaction of modules	Interaction and working of all modules as one	Software working per given specifications
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	Development team and End-users
Testing techniques	Usually Whitebox, and Greybox	Whitebox, and Blackbox	Usually Blackbox, and Greybox	Black-box testing
Automation	Automatable using JUnit, PHPUnit, TestNG, etc.	Automatable using Soap UI, Rest Client, etc.	Automatable using Webdriver	Automatable using Cucumber
Scaffolding	Complex (require drivers and/or stubs)	Moderate (may require drivers and/or stubs)	No drivers/stubs required	No drivers/stubs required

4. SOFTWARE TESTING TECHNIQUES

These are the various techniques that are used in testing software to ensure it performs as expected. Testing techniques specify the strategy used in developing test cases for conducting the testing and in analyzing test results [2] while increasing test coverage (since exhaustive testing is not possible) to achieve more effective testing. They help identify test conditions that are otherwise difficult to recognize. There are several testing techniques with each technique covering different aspects of the software to reveal its quality. Utilizing all the testing techniques in testing a given software is not possible, but the 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 techniques combined, the better the testing result, coverage, and quality [21]. There are three essential testing techniques [13]: White-box, Black-box, and Grey-box testing.

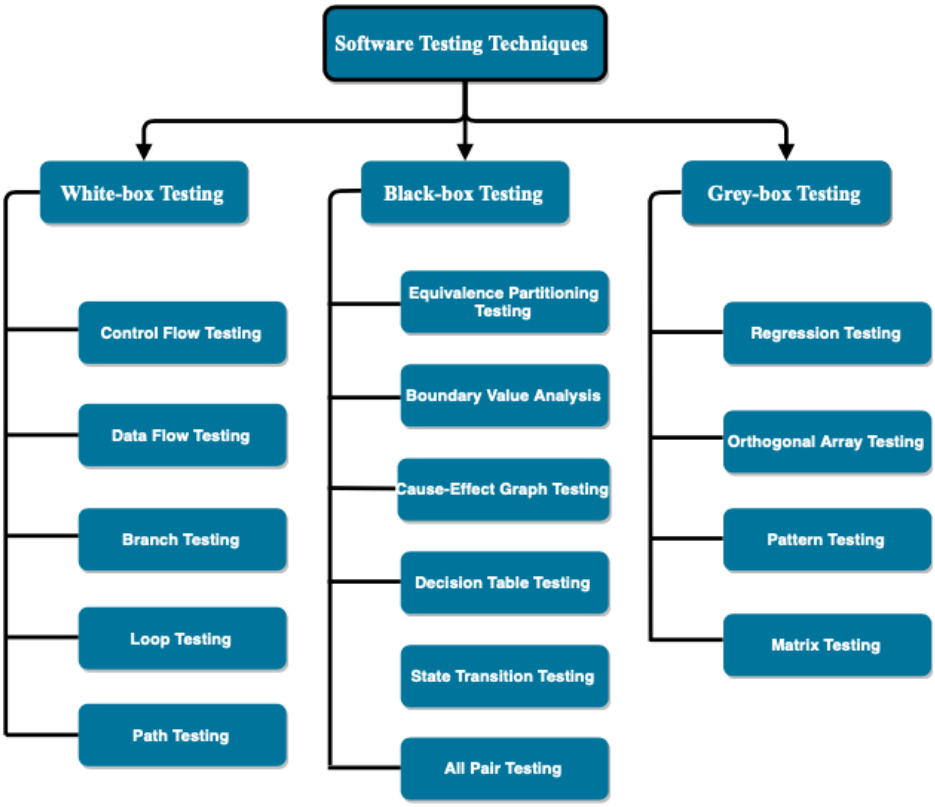

Figure 3: Software Testing Techniques

4.1. WHITE-BOX TESTING

This is a testing technique in which the internal structure and implementation of software being tested are known to the tester. In white-box testing, full knowledge of source code is required because test cases selection is grounded on implementation of the software entity; internal view of the system and tester's programming skills are used to design test cases [18]. Tester selects inputs to exercise program paths and compare the output with the expected output. White-box testing is also called Structural, Transparent Box, Glass Box, Clear Box, Logic Driven, Open Box Testing. White-box testing, although usually done at the unit level, is also performed at integration and system levels of the software testing process [13]. Some white-box testing types include: Control Flow, Data flow, Branch, Loop, Path Testing [13]. Some commonly used structural testing types are discussed below.

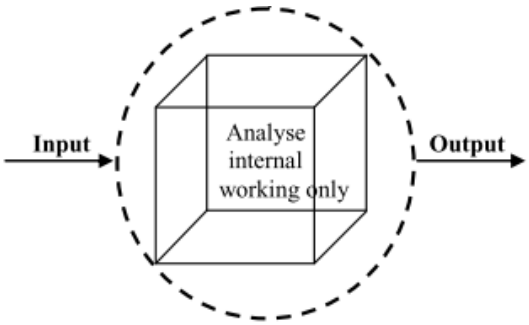

Figure 4: White-box Testing [22]

Table 2: Pros and Cons of White-box 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

SOME COMMON WHITE-BOX TESTING TYPES

4.1.1. CONTROL-FLOW TESTING

Control flow testing is a type of white-box testing in which control flow graph (CFG) paths, nodes, and conditions are selected, test cases are written for executing these paths, and each path, node or statements are traversed at least once to check the flow of control and determine the order of execution. By examining the control structure, the tester can select and design test cases [23]. Typically, a test case is an entire path from entry to exit nodes of the CFG. The selected set of paths is used to achieve a certain degree of testing thoroughness. Control-flow testing is most applicable to new software for unit testing [24].

A typical CFG of a program comprises of a set of nodes and edge, 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 have a 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) [25]. There are several conventions for flowgraph models with subtle differences (e.g., hierarchical CFGs, concurrent CFGs). Control-flow testing supports the following test coverage criteria [25]:

- *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 3: Pros and Cons of Control-Flow Testing

Advantages	**Disadvantages**
Catches 50% of all bugs caught during unit testing [24]	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 of white-box 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 [23]. 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 testing forms:: (1) define/use testing, uses some simple rules and test coverage metrics; (2) program slices - uses segments of a program [26]. Data flow testing uses the following Test Coverage Criteria in creating test cases for the test [23]:

- *All-defs (AD) coverage*: Has a path from every definition to at least one use of that definition
- *All-uses (AU) coverage*: For every use of a variable, there is atleast one path from the definition to its use.
- *All-c-uses (ACU) coverage*: 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*: For every variable, there is a path from each of its definition to each of its c-use. If there is any defined variable with no c-use following it, then p-use is considered.
- *All-p-uses (APU) coverage*: For every variable, there is a path from each of its definitions 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*: For every variable, there is a path from each of its definition to each of its p-use. If there is any defined variable with no p-use following it, then c-use is considered.
- *All-du-paths (ADUP) coverage*: For each def-use pair, all paths between definitions and uses must be covered. It is the strongest data-flow testing strategy since it is a superset of all other data flow testing strategies. Moreover, this strategy requires the greatest number of paths for testing.

Table 4: 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 [27].
Identify multiple variable declarations	Can have an infinite number of paths due to loops

4.2. BLACK-BOX TESTING

This is a software testing technique in which the internal structure/ implementation of software being tested is not known to the tester. It can be functional (such as integration testing) or non-functional (such as performance testing), though usually functional. Test cases are built around requirement specifications. In Black-box testing, the emphasis is given on evaluating fundamental aspects of software using thorough test cases, and generally, on maintaining the integrity of external information [13]. For a given test case, the tester verifies proper acceptance of inputs and correct production of outputs against test oracle. This testing can 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. Black-box testing is also called Opaque, Functional, Specification-based, Close-box, Behavioral, and Input-Output testing. Some Black-box testing types include: Equivalence Partitioning, Cause-Effect Graph, Fuzzing, Boundary Value Analysis, Decision Table, State Transition, Orthogonal Array, and All Pair Testing [22]. Some common black-box testing types are discussed below.

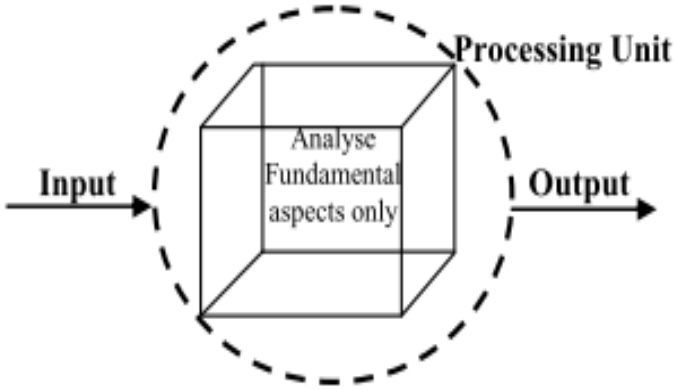

Figure 5: Black-box Testing [22]

Table 5: 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 clearly 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

SOME COMMON BLACK-BOX TESTING TYPES

4.2.1. EQUIVALENCE PARTITIONING TESTING (EP)

The testing technique of dividing the input domain of a program into different equivalence classes to reduce the number of test cases. One element from each equivalence class (EC) is then selected as test cases. This method is used to avoid test redundancy and give a sense of complete testing. EC Testing can be weak or strong. In Weak Equivalence Class Testing (WECT), the number of test cases is defined by chosen 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) is based on the cartesian product of partition class, i.e., testing all interactions of all equivalence classes [28].

Table 6: 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.	Cannot handle boundary value errors. Need to be supplemented by boundary value analysis

4.2.2. BOUNDARY VALUE ANALYSIS TESTING (BVA)

This is a black box test selection technique that aims at finding software errors at the boundaries of equivalence classes. Unlike the Equivalence Partitioning technique (uses only input domain), BVA uses both input and output domains in creating test cases. BVA complements EP in such that while EP selects tests from within equivalence classes, BVA focuses on tests at and near the boundaries of equivalence classes [28]. Tests derived using either of the two techniques may overlap.

Table 7: 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. GREY-BOX TESTING

Grey-box (translucent) testing technique that takes the straightforward technique of black-box testing and combines it with the code-targeted systems in white-box testing. Some knowledge of the internal working of the software is required (usually of the part to be tested) in designing tests at the black-box level. More understanding of the internals of software is required in grey-box testing than in black-box testing, but less compared to white box testing [13]. Gray box testing is much more effective in integration testing and is the best approach for functional or domain testing, also a perfect fit for Web-based applications [29]. Some grey-box testing types include: Orthogonal Array, Regression, Pattern, and Matrix Testing. Some of these testing are discussed.

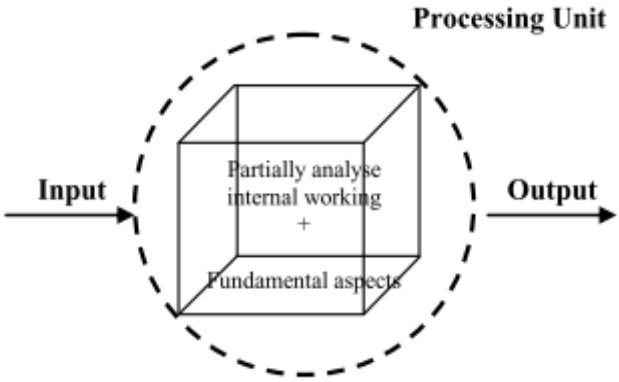

Figure 6: Grey-box Testing [22]

Table 8: 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	Gray box testing is not suitable for algorithm testing.

SOME COMMON GREY-BOX 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 is not negatively affected by the modification. 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) [30].

Table 9: 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 type of testing that uses pair-wise combinations of data or entities as test input parameters to increase the scope. The selected pairs of parameters should be independent of one another. 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 [31].

Table 10: 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.	

COMPARISON OF SOFTWARE TESTING TECHNIQUES

There is no one particular technique that is better, however, depending on the testing requirements and needs one technique can have some advantages over others and vice. In testing any software, exploring and combining many testing techniques helps in eliminating more bugs thereby increasing the overall quality of the software than sticking to one technique. The table below presents comparisons of the three discussed testing techniques using some criteria.

Table 11: Comparison of Testing Techniques

Criteria	**White-box**	**Black-box**	**Grey-box**
Required knowledge	Full knowledge of the internal working of the software.	Knowledge of the internal working of software is not required.	Limited knowledge of the internal workings of the software.
Performed by	Usually testers and developers.	End-users, developers, and testers	End-users, developers, and testers
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.
Granularity	High	Low	Medium
Time consumption	Very exhaustive and time-consuming	Exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.
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
Algorithm testing	Suitable for testing algorithms.	Unsuitable for testing algorithms.	Inappropriate for testing algorithms.
Also known as	Transparent-box, Open-box, Logic-driven, or code-based testing.	Closed-box, data-driven, functional, or Specification-based testing.	Translucent testing

5. SOFTWARE TESTING TYPES

***Testing Types*:** are the various testing that are performed at a particular test level based on a proper test technique to address testing requirements in the most effective manner [12]. There are many types of testing each serving different purposes. In a survey conducted by the International Software Testing Qualifications Board (ISTQB) [32], some of the most important types of testing are:

Table 12: Software Testing Types

Testing Type	Object	Technique Type	Testing Level
Functional Testing	Test functions of a software	Blackbox testing	Acceptance and System level
Performance Testing	Testing software responsiveness and stability under a particular workload	Blackbox testing	Any level
Security Testing	Protect data and maintain software functionality	Whitebox testing	Any Level
Usability Testing	Check ease of use of software	Blackbox testing	Acceptance and System level
Use case Testing	Checking that path used by user is working as intended	Blackbox testing	Acceptance, System and Integration level
Exploratory Testing	Validate experience of user	Ad-hoc testing	Acceptance and System level

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, some important software testing concepts, their advantages, and disadvantages are discussed, comparisons of software testing techniques and software testing levels are presented. Learning about and successful usage of these software testing methods in software development will help testers carry out software testing in a more effective manner thereby improving software quality.

REFERENCES

[1] A. Dennis, B. H. Wixom, and D. Tegarden, *Systems Analysis and Design with OOP Approach with UML 2.0*, 4th Editio. USA: John Wiley & Sons, Inc., 2009.

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

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

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

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

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

[7] S. Rogerson, 'The Chinook Helicopter Disaster', 2002. [Online]. Available: https://www5.in.tum.de/~huckle/chinook_software.pdf.

[8] N. I. of S. andTechnology (NIST), 'Software Errors Cost U.S. Economy $59.5 Billion Annually: NIST Assesses Technical Needs of Industry to Improve Software-Testing', *Web.archive.org*, 2002. [Online]. Available: https://web.archive.org/web/20090610052743/http://www.nist.gov/public_affairs/releases/n02-10.htm. [Accessed: 11-May-2019].

[9] C. Jones, 'Software Quality in 2012: a Survey of the State of the Art', 2012.

[10] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

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

[12] Altexsoft, 'Quality Assurance - Quality Control and Testing: The Basics of Software Quality Management', Kharkiv, Ukraine, 2016.

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

[14] K. Sneha and G. M. Malle, 'Research on software testing techniques and software automation testing tools', *2017 Int. Conf. Energy, Commun. Data Anal. Soft Comput. ICECDS 2017*, pp. 77–81, 2017.

[15] M. A. Jamil, M. Arif, N. Sham, A. Abubakar, and A. Ahmad, 'Software Testing Techniques : A Literature Review', no. November, 2016.

[16] P. Borba, *Testing techniques in software engineering : Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures*. Springer-Verlag, 2010.

[17] C. Padmini, '1- Beginners Guide To Software Testing', pp. 1–41, 2013.
[18] 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.
[19] 'Software Testing - Wikipedia'. [Online]. Available: https://en.wikipedia.org/wiki/Software_testing. [Accessed: 05-May-2019].
[20] R. V. Binder, *Testing Object-Oriented Systems: Objects, Patterns, and Tools*. Addison-Wesley Professional, 1999.
[21] I. Jovanovic, 'Software Testing Methods and Techniques', *IPSI BgD Trans. Internet Res.*, vol. 5, no. 1, pp. 30–41, 2009.
[22] Mohd. Ehmer Khan and Farmeena 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.
[23] J. Badlaney, R. Ghatol, and R. Jadhwani, 'An Introduction to Data-Flow Testing', *Control*, pp. 1–8, 2006.
[24] 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].
[25] N.-W. Lin, 'Software Testing (CS5812) - Control Flow Testing'. [Online]. Available: https://www.cs.ccu.edu.tw/~naiwei/cs5812/st4.pdf.
[26] M. New, 'Data Flow Testing Swansea University UK'.
[27] T. Su *et al.*, *A Survey on Data-Flow Testing*, vol. 50, no. 1. 2017.
[28] 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].
[29] 'Software Testing Class - Grey box'. [Online]. Available: https://www.softwaretestingclass.com/gray-box-testing/.
[30] 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.
[31] Alex Samurin, 'Explore the World of Gray Box Testing', 2003. [Online]. Available: http://extremesoftwaretesting.com/Articles/WorldofGrayBoxTesting.html. [Accessed: 19-May-2019].
[32] ISTQB, 'Worldwide Software Testing Practices Report'.
International Software Testing Qualifications Board (ISTQB) (2017). Worldwide Software Testing Practices Report. [online] ISTQB, pp.1-40. Available at: https://www.istqb.org/documents/ISTQB 2017-18_Revised.pdf [Accessed 21 May 2019].