Available online at www.garmian.edu.krd

Passer Journal of Basic and Applied Sciences

Passer 7 (Issue 2) (2025) 1001-1011

Original Article

Black Box Software Testing Techniques: a Literature Review

Zainab Momen Alshamaa*[1], Nada Nimat Saleem[1]

[1]*Department of Software Engineering, College of Computer Science and Mathematics, University of Mosul ,Mosul, Iraq.*

Received 22 February 2025; revised 11 August 2025;
accepted 25 August 2025; available online 21 September 2025

DOI: 10.24271/PSR.2025.508216.1963

ABSTRACT

There are numerous approaches for generating test cases, with many depending on knowledge of the internal workings of the software. With regard to generating software test cases, their internal knowledge is not always required. This is because there are various approaches for generating test cases. This 'black box testing' is perhaps one of the most famous approaches, as it only works with a system's outputs and inputs and does not take internal code or structure into account. This paper aims to analyze the aspects of black box testing approaches and their use in different software systems. One important advantage of black box testing is that it can be done even by people bereft of the programming knowledge. Each technique within this category has its own advantages and depends on particular system requirements which determine changing test systems and specifications. Understanding the requirements and goals of testing is pivotal for choosing the right technique. Increase test coverage while decreasing test case redundancy and fault findability are primary goals of these approaches. In this review, over 20 studies were analyzed to understand how these techniques are applied in practice. The evaluation of the techniques is categorized based on three main criteria: the effort required for testing, the efficiency of the testing process, and the overall effectiveness in fault detection. Ultimately, the findings suggest that no single technique is universally optimal; rather, testers must make context-based decisions to select the most appropriate method for their needs.

Keywords: Black box technique, Software testing, Quality assurance, Equivalence Class Partitioning, Decision Table.

1 Introduction

Software quality assurance is essential and impacts software success. Software performance and functionality are direct outcomes of QA, and QA is an integral part of SDLC. Quality is measured against user requirements and expectations. QA guarantees the performance of the software under test is met, as well as other factors, such as functionality, usability, reliability, maintainability, and post-test verification. Various methods of software testing include black box, white box, and gray box testing [1]. Black box testing is one of the most significant testing methods in software testing used to identify the conditions under which software fails to perform according to its specifications [2]. Similarly, it is also capable of specifying the creation of test cases without any information regarding the internal structure of the software. Since black box testing is directly or indirectly derived from the software specification, it is also referred to as specification-based testing [3]. Unit, integration, and system testing are some of the numerous levels of testing that can be applied in black box testing. To facilitate effective testing of software products on the basis of continued development, the black box testing techniques of equivalence partitioning, boundary value analysis, decision table testing, and graph-based testing have emerged. Each technique has its own pros and cons. The tester must choose the most appropriate technique that is to be implemented on the system under test [4,5]. This paper is concerned with the investigation of these techniques and their implementation on various systems. Section 2 includes a description of the basic methods in software testing. Section 3 includes the definition of black box testing, how it works, commonly used techniques, and how each technique works. In Section 4, a literature review is conducted in order to provide a comprehensive view of black box testing techniques and create a strong knowledge base on the topic. In Section 5, techniques are compared from three different perspectives. In Section 6, a summary of black box testing techniques is provided.

* Corresponding author
E-mail address zaynab.23csp15@student.uomosul.edu.iq (Instructor).
Peer-reviewed under the responsibility of the University of Garmian.

2 Software Testing

Software testing techniques are usually divided into three broad categories: white-box testing, black-box testing, and gray-box testing. as seen in Figure 1[4]. Every technique has its own method, depending on the type of system and test objective. White-box testing initially analyzes the internal software architecture. The test demands complete knowledge about the source code and program structure. This type of testing is usually

used to test implementation logic, validate control paths, and verify the quality of the code in terms of efficiency and proper data flow. It is of special importance for the detection of logical errors or those caused by misallocation of variables or conditions. Different to this, black-box testing also does not call for any expertise in the inner code and solely focuses on what the system gives as output against specific inputs on the basis of specifications and function requirements. Black-box testing is used to examine the system for compliance with requisite functions, input and output processing, error reporting, and the interaction with a user. Black-box testing is widely applied for acceptance testing conducted by users or third-party groups since it picks up the system's behavior at the end user's perspective. The third type, gray-box testing, is a compromise with elements of both white-box and black-box testing. In this case, the tester has no access to the code but has some knowledge of the system architecture or technical information such that they can prepare tests that utilize this knowledge to maximize test coverage efficiency. This is used where accurate tracing of data or analysis of repeated faults is required which cannot be easily determined by either of the above two methods separately. These three approaches assist i dealing with different aspects of system quality. White testing is used to improve internal code, black testing is used to test external behavior, and gray testing is used to correct weaknesses because of poor visibility. The choice of the appropriate approach depends on the development stage, the type of the system, and the level of access allowed to the tester [6-7].

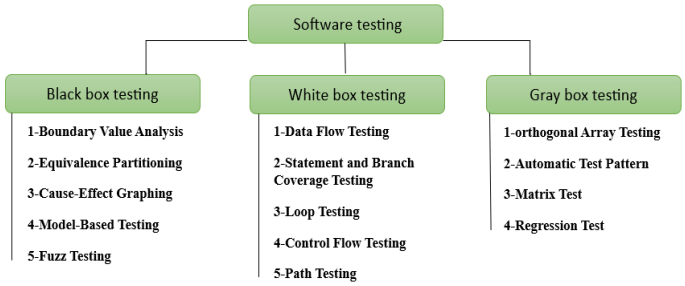


Figure 1: basic approaches to software testing.

3 Black Box Testing

Black box testing is a crucial method of software testing involving verification of the functionality of a system without consideration of the structure of its internal code or architecture. Black box testing is a type of test that is utilized to analyze the behavior of a system based on specific inputs and outputs in comparison to user requirements or approved specifications. The primary purpose of this method is to ensure that the software performs as it is supposed to do from the functionality standpoint, regardless of the internal design of those functions. The tester devises various scenarios for testing various inputs in an attempt to discover faults or undesirable behavior.

Black box testing is used at various stages of the test life cycle, such as system testing, acceptance testing, integration testing, and even unit testing if the test staff is unaware of the code. Through such testing, conformance of the system with functional requirements can be verified and desired outcomes achieved under varying conditions. In order to gain maximum efficiency for black box testing, a suite of systematic methodologies has been created to cover every possible scenario utilizing a minimal quantity of test cases. Amongst the most prominent of these processes is equivalence partitioning, a method of placing similar values into categories presumed to have identical results, minimizing the number of tests. Boundary value analysis is also used to test boundary values at the beginning and end of each input group, where the majority of errors will be encountered. Additionally, decision table testing is useful when checking complex business rules with many related conditions. Finally, graph-based testing methods are used in systems that are state transition or interface based. The system is represented in the form of a transition state diagram to analyze transition behavior and achieve maximum coverage. With these methods, comprehensive coverage of a system's functional aspects can be ensured with minimal effort to create test cases. Black-box testing is most suitable when the goal is to determine if a system conforms to external specifications without regard to how the system is built internally [8].

3.1 Equivalence Class Partitioning

The process of partitioning equivalence classes focuses on analyzing the input conditions explicitly or implicitly mentioned in the Software Requirements Specification (SRS). test cases are built from equivalence classes. Equivalence classes are a set of input conditions that are expected to behave in the same way as the test for each class. This approach streamlines the number of tests required while satisfying functional coverage. Take, for example, an application that includes a field for accepting numbers—more specifically, a numeric input. Classes are divided into valid and invalid. A test case can then be created that divides the input range into two equivalence classes: valid range and invalid range. This approach negates the need to test each input number individually. The following steps can be followed while implementing equivalence partitioning [9]:

Step 1: Analyze the specification to determine equivalence partitions.

Step 2: Create Test Coverage Items (TCIs) from the equivalence partitions.

Step 3: to work out data values to use for each TCI.

Step 4: Verify the test cases Verification consists of two steps: first, complete the TCI table, and then review your work.

3.2 Boundary Value Analysis

Boundary Value Analysis (BVA) is a most frequent software testing method used for finding potential errors by evaluating the inputs at the boundary of the acceptable ranges. It is based on the fact that system failures occur closer to the boundary values of input domains rather than their interior parts. Thus, BVA is concerned with the necessity to develop test cases that test the behavior of the system in processing the minimum and maximum valid inputs. To assist in ensuring the reliability of the software, this method generally tests a collection of values like the lowest valid input, the highest valid input, values just below and just above those limits, and possibly a typical normal value somewhere in between. This type of boundary testing is great for uncovering common errors that might not be caught by tests designed to target only average or mid-range values, The technique typically involves testing the following ranges of input values[8]:

1- Minimum valid inputs, or the lowest figures that the system should accept.
2- Maximum valid inputs, or the highest values that are acceptable to the input range.
3- Inputs just below the minimum, to determine if the system properly rejects values below the valid range.
4- Inputs just above the maximum, to determine if overflow or out-of-range errors are properly handled.
5- Exact boundary values, i.e., values equal to the minimum or maximum specified.

3.3 Decision Table

A decision table is a black-box technique and an effective method for analyzes the logic of relationships between inputs and outputs, particularly in those systems which have numerous conditions that is bound to influence decision making. This method uses a table which defines all possible conditions (inputs) and their relative actions (outputs). Each row of the table depicts a rule that represent a definite amalgamation of conditions that lead to a particular action. The columns contain possible values of each condition (yes/no, true/false) and the corresponding actions to be executed on those states. This method helps in system state coverage that improves software quality, find uncovered gaps, and uncover logical inconsistencies. In complex systems with multiple input interactions, this technique aids in documenting the system decision logic. Nonetheless, simple cases with few conditions may limit the applicability of the decision table. In such circumstances, preparing the table becomes an arduous task without any tangible benefit. Moreover, adding more conditions results in an exponential increase in the number of rules, making the decision table unwieldy. Therefore, it is recommended for use in cases requiring precise logical analysis and multiple conditional interaction[10,11]. Figure 2 shows the flowchart for generating test cases using decision table-based testing.

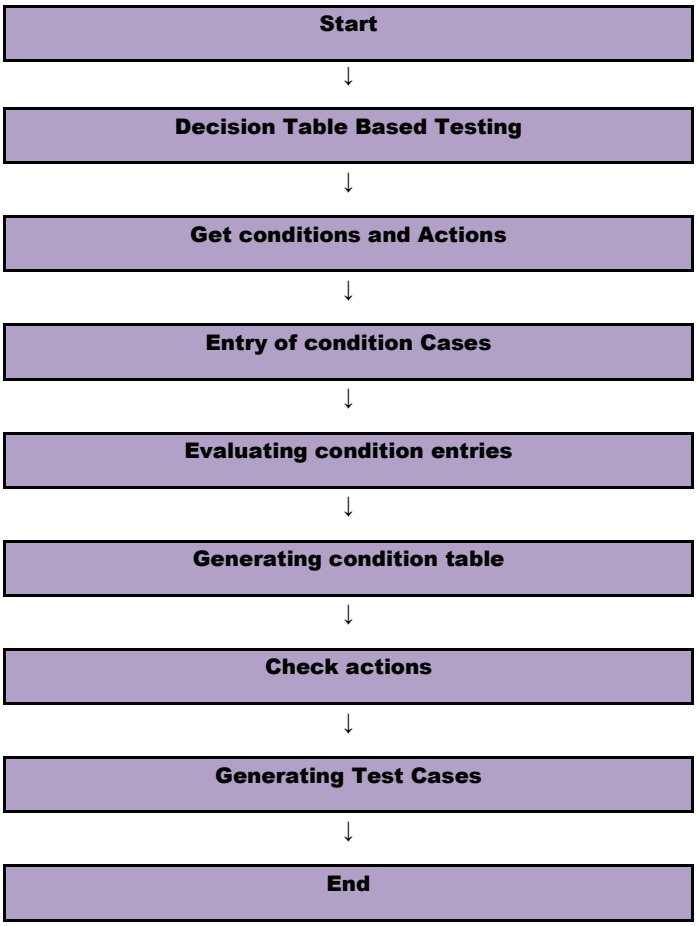


Figure 2: Flow chart for generating test cases using decision table-based testing.

3.4 Graph Based Testing

In the field of software testing, the cause and effect diagram testing strategy is used modeling the logical relationships between input conditions, known as causes, and resulting output conditions, known as effects. The Requirements of the Software is typically represented in a diagram that show how combinations of inputs are expected to produce given outputs. This diagram captures the logic of relations of causes to their outcomes in a formalized way and it is feasible to obtain logical expressions from it to formulate exact test cases. While this method originated from hardware testing, it has been modified for the purpose of software testing, especially for data entry and graphical form based components with multiple validations. Often these relations are graphically depicted, also referred to as graph based testing where the system is graphically represented and the objects or states within a system alongside their relationships mapped out. This enables the creation of comprehensive test scenarios that cover a wide range of possibilities. This explanation form captures complex system models with a high number of conditional parameter dependencies interacting with

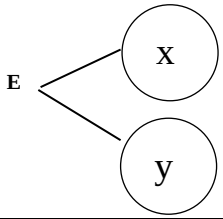
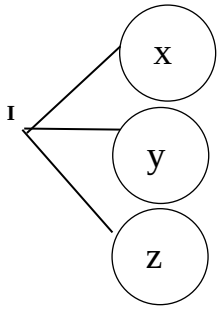
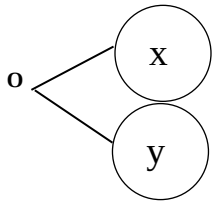
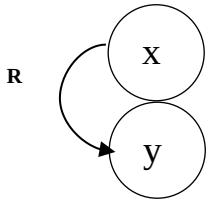
one another, although it may be less practical for simpler systems where such detailed modeling is unnecessary [2].

The diagram uses a set of symbols that represent a set of logical operations AND (∧), OR (∨), NOT (¬), as shown in Table 1, and apply a set of constraints to remove inapplicable test cases as shown in Table 2.[12]

Table 1: Notations for Cause Effect Graph.

Notation	Definition	For making e true
C → e	C implies e	C = 1 → e = 1
C ↛ e	C not implies e	C = 0 → e = 1
C1 ∧ C2 ∧ C3 → e	e when c1 and c2 and c3	C1 = 1, C2 = 1, C3 = 1 → e = 1
C1 ∨ C2 → e	e when c1 or c2	C1 = 0, C2 = 1 → e = 1 C1 = 1, C2 = 0 → e = 1 C1 = 1, C2 = 1 → e = 1

Table 2: Constraint symbol.

Constraint Symbol	Definition	Table
	The Exclusive (E) constraint specifies that only one of the specified conditions—such as X or Y—can be true at a time, but not both.	X | Y 0 | 0 0 | 1 1 | 0
	the Inclusive (I) constraint ensures that one at least of a set of conditions (e.g., X, Y, or Z) must be true, in such a way that it is impossible for all of them to be simultaneously false.	X | Y | Z 1 | 1 | 1 1 | 0 | 0 0 | 1 | 0 0 | 0 | 1
	The One and Only One (O) constraint also restricts the logic to have precisely one of the provided causes to be true—no more, no less.	X | Y 0 | 1 1 | 0
	Requires(R) constraint defines a dependency: if X condition holds, then Y condition must also hold. In this case, X cannot hold unless Y holds.	X | Y 1 | 1 0 | 1 0 | 0

3.5 State Transition Testing

State transition testing is an effective black box testing technique to test the behavior of a system that moves from one state to

another based on certain events or inputs. This becomes a powerful technique in systems where the output and next state depends not only on the current input but the current state of the system as well. Such systems include embedded systems, user authentication modules, workflow engines, and complex systems with intricate state logic. In state transition testing, a state transition diagram or table representing the valid states of the system, the transitions between these states, along with the inputs and outputs is created by the testers. Each state in the diagram represents a test case and the target is to validate that for every valid combination of the current state and input, the system moves to the desired next state with the correct response. This technique helps boundary identification by not only detecting incorrect state transitions but also discovering unimplemented or erroneous transitions bound to introduce system failure [8,13].

3.6 Use-Case Testing

Use case testing is a software testing technique that guarantees the system and the end-user requirements are matching. The testing is derived from certain scenarios that define how a user utilizes the system to achieve a specific goal. The character of use case testing is such that it is functionality-oriented from the user's perspective, not the system design or internal architecture perspective, and it is a form of black-box testing. Use case testing begins by identifying all significant use cases that signify the basic requirements of the system. Each use case is then analyzed, defining the steps of its interaction with the system, including inputs, outputs, preconditions, and expected results. Custom test cases are created for each use case, including input data, step-by-step information, and expected outputs. The cases are executed manually or with the help of automated tools in order to test the system's functionality in various usage scenarios. In this process, the defects which are not observable with separate functional or unit testing can be identified. Use case testing also serves to improve the user experience because it is derived from real-world user interaction with the system. This type of testing also identifies gaps in operational logic and unclear constraints in requirements. It is also easy to document and understand by developers, testers, and stakeholders. It is an important acceptance test, as it indicates whether the system is ready for actual use [8,14]

3.7 Pairwise Testing

pair testing is also known as all-pair testing, the most effective software testing strategy to minimize test cases with no loss of the level of test coverage. The main idea of the method is that the majority of the software bugs are the result of causing interaction between two input parameters and not between all inputs at once. Instead of trying all of those hundreds and often thousands of different combinations of the input, this involves only ensuring that each possible combination of distinct input values has been tried. Such black box testing techniques do not include having the system's internal implementation but only the behavior as defined due to the inputs. This is particularly useful where resources are scarce that prevent the implementation of all available test cases or it is a tight project schedule. The benefit of it is that one achieves greatly reduced test cases yielding fairly significant results, thus making it a very widespread practice in medium to large-sized software projects. Applications of pair testing are, but not restricted to, user interface testing, browser compatibility test, multi-device test, and highly advanced input systems. One of the limitations is that it doesn't guarantee the detection of defects resulting from three or four techniques of overlapping variables. Hence, sometimes it is recommended to implement more comprehensive coverage techniques when the risk factor is high [8].

3.8 Error Guessing

Guess-by-guess testing is a form of a manual testing method wherein the tester's expertise, intuition, and inventiveness are employed to detect potential defects in a system. This kind of testing is not governed by any rules in an organized manner of specific analytical steps. Rather, test cases are generated based on the tester's understanding of the system, experience with similar systems in the past, and his or her capability to think out of the box. As a type of testing, black-box testing is one that entails no knowledge of the source code details or internal implementation of the system. In such a procedure, the tester will conceive of the likely causes of a system failure, such as unpredictable data, maximum or minimum values, the same procedure carried out in an unusual manner, and a step that has been missed from the implementation sequence. For example, the tester may decide to attempt entering a negative number into a field that can only accept positive numbers, trying letters instead of numbers, or leaving some fields blank even if they are mandatory [8]

4 Related Work

this paper review the black box testing techniques used in previous studies, Irawan, Y., et al, applied the equivalence class division method in testing the banking information management system according to the input conditions included in the system specifications. The input conditions are classified into two classes: valid equivalence class and invalid equivalence class. The equivalence class consists of a set of inputs that share the same attributes, which allows one test case to be generated for each class, which greatly reduces the number of test cases and efficiently uses system resources. The researcher analyzed the set of input conditions to develop equivalence classes for them. The total number of successful test cases was 83.33%, while the number of failed test cases was 16.66% [15].

Researchers Ikhlaashi, S., et al, conducted a study to test the post-crossing program. In this study, two black box testing techniques were used: Partitioning Equivalence and Boundary Value Analysis to test the post-crossing program Postcrossing is an online project that allows members to receive real postcards instead of emails. Through interviews and note-taking, the functional requirements of the program were determined. In the equivalent partitioning method, three criteria were used and each

criterion was divided into a valid and invalid value, the total partitions would be 6, 5 successful cases and 1-failed case were obtained after testing In the Boundary Value method, only two criteria were used: the maximum number of address requests and the maximum number of cards sent to the same country because they represent ranges. The minimum and maximum can be determined, 6 partitions were created for each criterion, and three partitions, 5 successful cases and 1 failed case were obtained after testing [16].

In a study by Amrulloh et al. of Telkom Institute of Technology Purwokerto Software Engineering Department, the researchers applied black box testing employing the technique of equivalence partitioning to evaluate several of the most notable features in the Ternaku.id investment platform. The registration, login, and investment capabilities were the focus for testing to determine if the features met the functional requirements. 11 test cases were created and run: four to the registration process (with three successful results), five to test the login functionality (with three successes), and two to the investment feature (both successful) [17].

Researchers Zuhair et al. used black box testing by the equivalence class partitioning technique to assess the effectiveness of a decision support system for the selection of top performing employees. The system was tested with ten individual functional tests, two of which were login module tests, six save function tests, one edit option test, and one delete functionality test. The tests in most instances gave positive results, confirming proper system behavior. However, the edit function generated an error, indicating a glitch that needed close scrutiny [18].

Researchers Restu Langit, M. S., et al, conducted a study to test the website of the library system using the methods of boundary value analysis and equivalence division. This test is carried out on the main function of the website, which is the login and login menu. The test cases created using the equivalence division method are 26 test scenarios, 26 of which are successful and 0 are unsuccessful. For the boundary value analysis method, 6 test cases were created and executed successfully. It was found that the website-based library system, especially the functions of the registration menu and login menu, were in compliance with the specified requirements and provisions. The test results showed that no errors or malfunctions were detected in the system [19].

Researchers Maulana and Voutama examined the usability of black box testing specifically the employment of the equivalence partitioning approach in evaluating the functionality of the Population and Family Planning Department Karawang Regency official website (kbinten.id). With a test approach independent of knowledge of the system's source code, the scientists focused on the verification of user input of different pages and form fields for the website. By partitioning the inputs into equivalence classes, the researchers established a collection of 11 structured test cases that were tailored to be capable of distinguishing valid from invalid inputs. All the test scenarios which were executed proved successful, indicating that the system could process the expected inputs appropriately [20].

In another study, Hendri and others employed black box test concepts to validate a mosque management information system. The researchers adopted the equivalence class partitioning to ensure that functionalities of the software complied with expected user requirements. Without examining source code, the test cases were automatically created for testing input handling and system response. The output of the test process revealed that all the functions were performing as required without any errors being detected at the execution of the test cases [21].

Researcher Ardana, I. M. S conducted a study in which he discussed the use of two black box testing techniques, namely decision table and marginal value analysis. The study uses these two methods to evaluate the model. Periodic pension claim submission is part of employee pension fund management, and the model was selected for its importance and to ensure the integrity and quality of the calculations it produces. The program is tested. Test cases that do not pass the test require necessary modifications to the program to meet the business requirements [11].

Researchers Guo, X., et al conducted a test with the boundary value analysis approach to try out five programs. A new measurement by the term boundary coverage distance is presented. The BCD measure provides a foundation for testing the quality of test inputs in comparison to boundary coverage in boundary value analysis. This measure prioritizes measures by distance and lower and upper boundary consideration to measure how close test inputs to the limit yield test cases. Based on our experiments, the BCD-based method is able to generate test inputs close to the limit and may improve the fault detection ability of a random generated test set [22].

In a study conducted by Selviana et al., the researchers examined the performance and functionality of a community service information system developed for the Department of Community and Village Empowerment in Jombang District. They employed black box testing techniques, focusing specifically on Boundary Value Analysis (BVA) and the Cause-and-Effect Diagram method. Their findings revealed that the cause-and-effect approach was especially useful for evaluating features related to user interactions such as button operations and form submission functions—common elements in public service interfaces. On the other hand, the BVA technique proved effective in assessing fields that required specific input boundaries, such as names and address fields. The system's overall effectiveness score was calculated at 77.86%, indicating a satisfactory level of performance, though further enhancements were recommended to optimize functionality and user experience [23].

Meanwhile, Joosten et al. explored the Decision Table method as a software testing strategy in their investigation of a medical records system used within a veterinary hospital management application. This technique offered a structured way to represent conditions and actions, significantly reducing test complexity. Initially, 20 test cases were formulated to cover various scenarios within the system. However, by analyzing overlaps and consolidating similar test cases, the researchers successfully

minimized the number of necessary test cases to 13, without sacrificing coverage or accuracy. This demonstrates the efficiency of the decision table method in reducing both the volume of tests and the overall time required for software validation [24].

Anupama Y.K. and P.I. Khodanpour conducted a study to develop a program for generating test cases using decision table method. The program is implemented in C language. Based on the program, the tester enters the inputs, which are the number of conditions and condition instances. The next activity is to enter the actions. After that, the values of the condition table will be populated. The possible action instances are considered with respect to the condition table and test cases are generated for the values generated randomly by the program or as values entered by the tester [10].

The researcher Sianturi, E tested the program used to manage the pension funds of employees of the pension fund management company. From this program, a sample of a periodic pension benefit claim form was selected. Based on the workflow rules for data entry and model validation, a test scenario and data were created for use in the test. After passing the testing process, the results showed that there are some things that need to be improved in the sample model [25].

Researchers Handayanto, I. S., et al, conducted a study to test a story sharing application, which was built using the Kotlin programming language and tested using decision table technology. This test covers various features such as registration, login, and story uploading. The story uploading system test successfully covered a wide range of possible scenarios, and the results showed that the system behaved as expected [26].

Researchers Ehliman Krupalia, Sheila Bechirovic, Irvan Prazina, Emir Kojo, and Ingemar Pesic presented a software tool designed to generate cause-and-effect diagrams from system requirements. It is designed to facilitate the definition of nodes, logical relationships, and constraints in a cause-and-effect diagram, but the tool does not currently support the derivation of test cases. The tool was evaluated and the proposed tool proved its ability to handle diagrams of different sizes, and users found the tool easy to understand and provides clear outputs [27].

The test was conducted on a library website at Merdeka University in Malang to see if the system works well and meets user expectations. The test was conducted using black box testing with graph-based testing technology which describes a graph that represents the relationships between objects so that each object and its relationships can be tested. Four modules were tested on the website and the success rate was 62.5%. There are two modules that contain errors that need to be corrected in order to improve the functionality of the website as an information center so that it meets user expectations [28].

Researchers Putu Ayu Desi Angara Santi, Royana Afwani, Muhammad Ali Albar, Sri Endang Anggarwani, and Ahmad Zafarullah Mardiansyah from Mataram University conducted a study to test the Mataram University Academic Testing System developed by UPTpustic. Parity partitioning and boundary value analysis techniques were used to detect and fix defects and errors. Through interviews with uptpustik, test data was collected. Test cases were created for each of the four system modules (Student Module (Total Test Cases 127), Lecturer Module (Total Test Cases 52), Department Operator Module (Total Test Cases 88), and Faculty Operator Module (Total Test Cases 55). The test was conducted at Matram University for 3 days and 17 defects were found in Student Module, 10 defects in Lecturer Module, 32 defects in Department Operator, and 21 defects in Faculty Operator Module. Based on this, 80 bugs were found out of 332 test cases and 75.16% of the system test cases passed and 24.84% failed [29].

Researchers Cahyadi Pradipta, I. G. L. A., et al, conducted a study on the Access by KAI application using boundary value analysis and graph-based testing. In this study, Access by KAI, an application focused on the process of ordering train tickets online, was tested and the test result showed that one test result was appropriate (78%) and two test results were inappropriate (22%). This means that the black box test and graph-based test completely found the functional inconsistencies in the test case [2].

Researchers Sakinah, F. A., et al, from the Department of Information Engineering at the National Veterans University of Pembangunan in East Java conducted a study on the application of asset management using black box testing using two methods, marginal value analysis and equivalence partitioning techniques. The test results showed that out of 227 test scenarios, 226 scenarios met expectations. One bug was found where it could not display an error notification when the user entered an invalid value in the investment release form. The success rate of the test cases was 99 percent [14].

Researchers Ismail and Jalisal Effendi of the Informatics Management Study Program, AMIK Indonesia conducted a research study of implementation of source code bank programming using black box testing methods and three were utilized (graph-based testing, equivalent partitioning, and boundary value analysis). User needs satisfaction is good if the user unit value score is more than 0.80, the test results show user needs satisfaction of general users with a value of 0.90, user/member records with a value of 1.00, students with a value of 0.90, lecturers with a value of 0.82, and administrators with a value of 0.84 falls into the category of good since the value score of each unit of user is greater than 0.8 [30].

The Al-Irsyad University Cilacap lecturer team (Maryanti, D., et al) has created the Digital Counseling HIV/AIDS (D-Cohiva) site. D-Cohiva provides computer-assisted counseling for HIV/AIDS. The purpose of this study is to analyze the inputs and outputs of significance and verify if the current situations are appropriate to the expected situations. Black box testing was performed by utilizing state transition technique. There were eight test cases executed and all of them were 100% successful [31].

A research team Purwanti, S., et al,conducted a study on the testing of interactive multimedia application 'I Learn to Read' (ABACA). For ensuring that the multimedia application communicates as required by the user and to ensure the quality of the application, the testing has to be done. The testing is done by using black box testing using the state transition method. Twenty-three functions were tested and efficiently displayed the transitions depending on the functions available [32].

Setiawan, A. et al. conducted an in-depth study to test and evaluate medical devices inventory system of South Tangerang General Hospital, Indonesia. State transition method was used, the state transition technique is performed by considering the suitability of the flow from the current path to the subsequent one. Test outcomes on medical devices inventory system in South Tangerang General Hospital were 100% appropriate [13]. Table 3 shows a summary of previous studies.

Table 3: Represent the summarization of these previous works.

ID	Researcher's name	Technique name	Application domain	Percentage of Successful Tests	Limitations
1	Irawan, Y., et al. (2018)[11]	Equivalence Class Partitioning	banking information management system	83.33%	-It is not mentioned how to collect the system requirements. -The success rate of the tested system is not provided
2	Ikhlaashi, S., et al. (2020)[13]	Boundary Value Analysis And Equivalence Partitioning	postcrossing program	-	The success rate of the tested system is not provided.
3	Amrulloh, A., et al. (2023)[24]	equivalence partitioning	Ternaku.id Website	-	-It is not mentioned how to collect the system requirements - The success rate of the tested system is not provided
4	Zuhair, A., et al. (2020)[12]	equivalence partitioning	decision support system for selecting the best employees	-	-The success rate of the tested system is not provided
5	Restu Langit, M. S., et al. (2024)[25]	boundary value analysis and equivalence partitioning	website of the library system	-	It is not mentioned how to collect the system requirements
6	Maulana, T., & Voutama, A. (2023)[20]	equivalence partitioning	official website of the Population and Family Planning Department in Karawang Regency	-	- It is not mentioned how to collect the system requirements - The practical implementation of the method is unclear
7	Hendri, et al. (2020)[26]	equivalence partitioning	mosque management information system	100%	It is not mentioned how to collect the system requirements
8	Ardana, I. M. S. (2020)[29]	decision table and boundary value analysis	Periodic pension claim submission	-	nothing
9	Guo, X., et al. (2024)[15]	boundary value analysis	1findMiddle program 2 English program 3- triType program 4- nextDate program 5- miniSAT program	-	nothing

10	Selviana, E. Z., et al. (2023)[10]	Boundary Value Analysis and Cause Effect Graph	community service information system in the community and village empowerment service in Jombang District	77.86%	- It is not mentioned how to collect the system requirements - The practical implementation of the methods is unclear
11	Joosten, J., et al. (2020)[16]	decision table	medical records application program on a veterinary hospital management system	-	It is not mentioned how to collect the system requirements
12	Sianturi, E. (2020)[17]	Boundary Value Analysis and Decision Table	program used to manage the pension funds of employees of the pension fund management company	-	The practical implementation of the Decision Table Testing is unclear
13	Handayanto, I. S., et al. (2024)[18]	Decision Table	story sharing application	100%	It is not mentioned how to collect the system requirements
14	Santi, P. A. D. A., et al. (2022)[14]	Equivalence Partitioning and Boundary Value Analysis	Academic Information System of Mataram University	75.16%	nothing
15	Cahyadi Pradipta, I. G. L. A., et al. (2024)[2]	Boundary Value Analysis And Graph-Based Testing	Access by KAI application	78%	nothing
16	Sakinah, F. A., et al. (2024)[7]	boundary value analysis and equivalence partitioning	application of asset management	99%	nothing
17	Ismail, & Efendi, J. (2020)[27]	graph-based testing, equivalent partitioning, and boundary value analysis	application of source code bank programming		.No details of practical implementation were mentioned.
18	Maryanti, D., et al. (2023)[21].	State Transition Technique	D-Cohiva website	100%	.nothing
19	Purwanti, S., et al. (2024)[22]	State Transition Technique	interactive multimedia application 'I Learn to Read' (ABACA)	100%	. It is not mentioned how to collect the system requirements
20	Setiawan, A., et al. (2022)[23]	State Transition Technique	Medical Equipment Inventory	100%	. It is not mentioned how to collect the system requirements

5 Discussion

after reviewing the previous papers, the authors see that Black box testing techniques can be compared from three perspectives Testing effort, Testing efficiency and Test effectiveness. Test effort can be divided into two types: test case identification effort and test case execution effort. There is a trade-off between the two types. The effort required to identify test cases is the lowest in BVA and the highest in decision tables, while the effort

required to execute the test case is the lowest in decision tables and the highest in BVA.

Test efficiency and effectiveness depend on two measures: detecting errors in the system and non-repeated test cases. No technique can be said to be more efficient and effective than another technique. To allow the tester to analyze the specifications and test the appropriate technique, the tester must decide what is appropriate in the given situation - to use only one method or to use a combination of several methods.

The three techniques (equivalence class partitioning (ECP), boundary value analysis (BVA), and decision table (DT)) were compared to determine the most effective technique through testing the Lars website, a website developed by PT Andhara Prima Creative (PT APIK) that is used to assist in the hospital accreditation process based on applicable laws and regulations. The test results using the ECP technique showed that the test case failure rate was 51.8%, 33.3% for the BVA technique, and 46% for the DT technique. Based on these results, the Lars website needs to improve its functionality because there are still many errors. The most effective technique to use on the Lars website is ECP, which has a 51.8% error finding success rate and the lowest number of test cases compared to BVA and DT [1].

In another study, BVC was found to be more effective and less expensive than ECP. Therefore, no method is suitable in all situations and the tester must think about what is appropriate in the given situation [33,34].

In Boundary Value Analysis (BVA), the number of test cases produced will be much greater than in decision table-based testing—often very close to almost five times as many. This is due to the fact that the technique targets testing input values precisely at their boundary conditions, where defects will most likely occur. BVA targets testing the boundaries of input domains (minimum, maximum, and near-edge values), which leads to a larger number of test cases around these critical points. In contrast, when using Equivalence Class Partitioning (ECP), the number of test cases is raised moderately approximately 1.5 times higher than test cases produced by decision table methods. This is because ECP is the activity of selecting input and output data that behave in the same manner, putting them in logical classes, and selecting representative values from each class to be tested. This eliminates redundancy but still allows an efficient testing of each class of inputs. Hence, while decision tables focus on effectively covering rule-based combinations, BVA and ECP extend the test scope by targeting edge values or packaging similar data sets, respectively [8,11].

Conclusions

Software testing is a key activity in the software development cycle because it has an important function in delivering reliable and high-quality programs with lesser development effort and cost. One of the widely used techniques is black box testing, which is focused on testing the external functioning of the program without requiring the tester to understand the internal structure of the code or the implementation aspects. Its main goal is to verify whether the system is working as intended by its functional requirements and as expected by users. Selection of a proper black box testing method relies on the type of the application to be tested and also on the approach and intent of the tester. According to available literature describing black box techniques, several methods have been reported helpful for coping with different test scenarios. Techniques such as Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), and Decision Table Testing (DT) are predominantly applied, with each offering some advantages depending on the structure and complexity of the software system. The methods aid in systematic test case generation and enhance test coverage and fault detection.

First, the impact of black box techniques on improving testing effectiveness depends largely on the nature of the system being tested. In systems with clearly defined boundaries, Boundary Value Analysis is more effective. Meanwhile, in systems with a variety of inputs, Equivalence Class Partitioning is more suitable.

Second, comparing the effort and effectiveness of these techniques reveals an important balance between the number of test cases and their efficiency. Techniques like Boundary Value Analysis may require more effort in execution, while techniques such as Decision Tables offer accurate results with less effort in certain cases.

Finally, using a hybrid or combined approach of black box testing techniques can be the optimal choice in some cases, especially in complex systems that are difficult to test using a single technique.

References

[1] Putri, S. J., Putri, D. G. P. & Putra, W. H. N. Analisis komparasi pada teknik black box testing (studi kasus: website Lars). J. *Internet Softw. Eng.* **5**, 23 (2024).

[2] Kusuma, I. D., Pradipta, I. G. L. A. O. C. & Yuhana, U. L. Black box testing in the Access by KAI application using boundary value analysis and graph-based testing. *Infotech J.* **10**, 122–127, doi:10.31949/infotech.v10i1.9577 (2024).

[3] Khan, M. E. Different approaches to black box testing technique for finding errors. *Int. J. Softw. Eng. Appl.* **2**, 31–40, doi:10.5121/ijsea.2011.2404 (2011).

[4] Khaleel, S. I. & Salih, L. F. A survey of predicting software reliability using machine learning methods. *J. Softw. Eng. Appl.* **16**, 240–255, doi:10.4236/jsea.2023.165015 (2023).

[5] Khaleel, S. I. & Khaled, R. W. Selection and prioritization of test cases by using bees colony. *J. Softw. Eng. Appl.* **3**, 955–963, doi:10.4236/jsea.2013.312126 (2013).

[6] Alsarraj, R. G., Altaie, A. M. & Majeed, E. Z. Developing an automated model-based software testing tool from the design phase. *IEEE Access* **13**, 58548–58558, doi:10.1109/ACCESS.2025.3553967 (2025).

[7] Alsarraj, R. G., Altaie, A. M. & Ahmed, A. H. Constructing a software requirements tool based on the reusability attribute. *IEEE Access* **12**, 70017–70024, doi:10.1109/ACCESS.2024.3402144 (2024).

[8] Shen, J. J. *Software Testing: Techniques, Principles, and Practices*. Amazon Digital Services LLC – KDP, Seattle. (2019).

[9] Bierig, R., Brown, S., Galván, E., & Timoney, J. *Essentials of Software Testing*. Cambridge University Press, Cambridge. (2021).

[10] K., A. Y. B. I. Decision-Table Based Testing. *Int. J. Recent Innov. Trends Comput. Commun.* **3**, 1298–1301, (2015).

[11] Ardana, I. M. S. Pengujian software menggunakan metode boundary value analysis dan decision table testing. *J. Teknol. Inform. ESIT* **14**(3), 40 (2019).
[12] Agrawal, M., & Badhera, U. Approach for Minimization of Test Cases from Decision Table Generated from Cause Effect Graph. *Int. J. Comput. Appl.* **172**, 7–10, doi:10.5120/ijca2017915175, (2017).
[13] Setiawan, A. et al. Black box testing dengan teknik state transition testing pada inventori alat-alat medis. *J. Sains Teknol. (JSIT)* **2**, 151–158, doi:10.47233/jsit.v2i3.218 (2022).
[14] Sakinah, F. A., Aditiawan, F. P. & Nurlaili, A. L. Pengujian pada aplikasi manajemen aset menggunakan black box testing. *JATI* (*J. Mahasiswa Tek. Inform.*) **8**, 2814–2823, doi:10.36040/jati.v8i3.9524 (2024).
[15] Irawan, Y., Muzid, S., Susanti, N. & Setiawan, R. System testing using black box testing equivalence partitioning. *In Proc. 1st Int. Conf. Computer Science and Engineering Technology*, Universitas Muria Kudus, doi:10.4108/eai.24-10-2018.2280526 (2018).
[16] Ikhlaashi, S. & Putro, H. P. Komparasi dua teknik black box testing: equivalence partitioning dan boundary value analysis. *Annu. Res. Semin. (ARS)* **5**, 213–220 (2020).
[17] Amrulloh, A., Septiadi, A. D., Septiara, M., & Wicaksono, P. A. Black Box Testing Using the Equivalence Partitions Technique to Test the Functionality of the Ternaku.id Website. *J. Multimedia Trend Technol.* **2**, 171–178, (2023).
[18] Zuhair, A., Khadafi, F., Andriansyah, A. M., Saputra, B. & Saifudin, A. Teknik pengujian equivalence partions untuk pengujian aplikasi sistem penunjang keputusan pegawai terbaik menggunakan black box. *J. Teknol. Sist. Inf. Apl*. **3**, 132, doi:10.32493/jtsi.v3i3.5365 (2020).
[19] Langit, M. S. R., Voutama, A. & Ridha, A. A. Black box testing pada website sistem perpustakaan menggunakan metode equivalence partitioning dan analisis boundary value. *STRING* **8**, 355, doi:10.30998/string.v8i3.19529 (2024).
[20] Maulana, T. & Voutama, A. A black box testing dengan teknik equivalence partitions pada website Dinas Pengendalian Penduduk dan Keluarga Berencana Karawang. *Simika* **6**, 112–121, doi:10.47080/simika.v6i2.2536 (2023).
[21] Hendri, H. et al. Pengujian black box pada aplikasi sistem informasi pengelolaan masjid menggunakan teknik equivalence partitions. *J. Teknol. Sist. Inf. Apl*. **3**, 107, doi:10.32493/jtsi.v3i2.4694 (2020).
[22] [22] Guo, X., Okamura, H., & Dohi, T. Optimal Test Case Generation for Boundary Value Analysis. *Softw. Qual. J.* **32**, 543–566, doi:10.1007/s11219-023-09659-9, (2024).
[23] Selviana, E. Z., Wahanani, H. E. & Aditiawan, F. P. Black box testing in community service systems using boundary value analysis and cause effect graph methods. *East Asian J. Multidiscip. Res*. **2**, 4161–4184, doi:10.55927/eajmr.v2i10.6435 (2023).
[24] Joosten, J., Permanasari, A. E. & Adji, T. B. The use of decision table for reducing complex rules in software testing. *IOP Conf. Ser.: Mater. Sci. Eng.* **732**, doi:10.1088/1757-899x/732/1/012086 (2020).
[25] Sianturi, E. Boundary value analysis and decision table testing methods in software testing. *Int. J. Info. Tech. Educ.* **1**, 124–126 (2022).
[26] Handayanto, I. S. & Nuryasin, I. Pengujian blackbox decision table pada sistem aplikasi mobile sharing story app. *Smart Comp.* **13**(2), 383–394, doi:10.30591/smartcomp.v13i2.6572 (2024).
[27] Krupalija, E. New graphical software tool for creating cause-effect graph specifications. *J. Commun. Softw. Syst.* **18**, 311–322, doi:10.24138/jcomss-2022-0076 (2022).
[28] Purwaningtyas, E. & Jatmiko, A. R. Pengujian Black Box Website Perpustakaan Universitas Merdeka Malang Berbasis Graph Based Testing. *J. Ilmiah Comput. Insight* **6**(1), doi:10.30651/comp_insight.v6i1.16862, (2024).
[29] Santi, P. A. D. A., Afwani, R., Albar, M. A., Anjarwani, S. E. & Mardiansyah, A. Z. Black box testing with equivalence partitioning and boundary value analysis methods. In *Proc. Academic Information System* of Mataram University, 207–219, doi:10.2991/978-94-6463-084-8_19 (2022).
[30] Ismail, I. & Efendi, J. Black-box testing: analisis kualitas aplikasi Source Code Bank Programming. *J. JTIK* **4**, 1, doi:10.35870/jtik.v5i1.148 (2020).
[31] Maryanti, D., Pangesti, A. R. & Suprihatiningsih, T. Black box testing for HIV AIDS digital counseling website (D-Cohiva Apps) with state transition technique. *J. Penelit. Didik. IPA* **9**, 822–827, doi:10.29303/jppipa.v9ispecialissue.6087 (2023).
[32] Purwanti, S. et al. State transition method analysis for testing the interactive multimedia applications. *PIKSEL* **12**, 89–96, doi:10.33558/piksel.v12i1.8521 (2024).
[33] Kumawat, N., Sharma, Y. & Pilania, U. Comparative study between equivalence class partitioning and boundary value analysis testing methods. *Int. J. Innov. Res. Technol*. **2**, 74 (2016).
[34] Shahbaa, I. & Al Thanoon, A. A. Design a tool for generating test cases using swarm intelligence. *J. Comput. Math.* **1**, 421 (2013).