Systematic Literature Review on Black-Box Testing

Diaconu Ana-Maria, Drăghiciu Diana and Duma Amalia-Diana

ARTICLE INFO

Keywords:
black-box testing, system verification, error guessing, test case selection, testing framework, test automation

ABSTRACT

Black Box Testing is a critical software testing technique that focuses on evaluating the behavior of a software system without considering its internal structure. This technique has become increasingly important as software systems become more complex and interconnected. In this Systematic Literature Review (SLR), we present an overview of the state-of-the-art in Black Box Testing. Our objective was to identify and analyze the most relevant studies in this field and provide a comprehensive summary of their findings. We searched three major academic databases (IEEE Library, ACM Library, and Google Scholar) and included a total of 10 studies in our review. Our analysis revealed several emerging trends in Black Box Testing, including the use of machine learning techniques, model-based testing, and test tailoring and enhancement. We also identified several challenges and limitations of current Black Box Testing approaches, such as the difficulty of generating effective test cases for complex systems. Overall, this SLR provides a comprehensive overview of the current state-of-the-art in Black Box Testing and can serve as a valuable resource for software developers, testers, and researchers who are interested in this topic.

1. Systematic literature review (SLR)

Black Box Testing is a software testing technique that focuses on the behavior of a software system without considering its internal structure. This technique is widely used in industry to evaluate the quality of software systems and ensure that they meet the requirements specified by the stakeholders. As software systems become increasingly complex and interconnected, the need for effective Black Box Testing techniques becomes more crucial.

In recent years, several researchers have published studies that propose new methods and tools to improve Black Box Testing. However, due to the large and growing body of literature in this field, it can be challenging to keep up with the latest advancements and identify the most relevant studies.

To address this challenge, we conducted a Systematic Literature Review (SLR) to provide an up-to-date overview of the state-of-the-art in Black Box Testing. Our objective was to identify and analyze the most relevant studies in this field and provide a comprehensive summary of their findings. This SLR includes 10 studies published between 2002 and 2021 and covers a broad range of topics related to Black Box Testing, including techniques, tools, and frameworks. The findings of this SLR can be useful for software developers, testers, and researchers who are interested in Black Box Testing. By synthesizing the existing knowledge in this field, we aim to provide a solid foundation for future research and development of effective Black Box Testing techniques.

2. Study design

2.1. Review need identification

Black box testing is a technique used in software testing where the tester does not have access to the internal workings of the system being tested. Our work aims to identify and analyze the existing literature on black box testing. By doing so, this review aims to provide a comprehensive understanding of the current state of knowledge in this area, identify research gaps, and provide directions for future research. The motivation that drives us to conduct this SLR is given by the following reasons:

1. Testing is a crucial aspect of software development that ensures the software conforms to the desired quality standards, especially given the rising complexity of software. Black box testing is particularly important in ensuring that the software functions as expected in different scenarios and use cases.
2. Black box testing has been studied extensively in the past, and a significant amount of research has been conducted in this area. However, due to the vast amount of literature available, it can be challenging to identify

ORCID(s):

the most relevant studies and to synthesize the findings. This review aims to address this gap by providing a comprehensive analysis of the existing literature on black box testing.

2.2. Research questions definition

Our systematic literature review (SLR) aims to achieve its goal by focusing on two key research questions. Each research question is designed to address a specific objective, which are as follows:

- RQ1- *What approaches have been developed to support Black-Box Testing?* By answering this question, our objective is to provide a comprehensive overview of the different methods and techniques that have been proposed in the literature to support black-box testing.
- RQ2- *How have Black-Box Testing approaches been evaluated and what were the reached conclusions?* The answer to this question means to provide insights into the effectiveness of different black-box testing approaches, based on empirical evidence from studies that have applied these methods in practice.

By addressing these two research questions, we can provide a comprehensive overview of the current state of black-box testing research, highlight the strengths and limitations of existing approaches, and identify potential areas for future research and development.

2.3. Protocol definition

The steps and the rules for conducting the SLR are defined during this activity. We define five important steps, whose details are described in the following sections. The phases that we followed to conduct the SLR are:

- the resources and search terms used to search for primary studies,
- the selection process used to include or exclude studies from the systematic review,
- the data extraction strategy, which describes how to obtain the required information from each primary study,
- the data synthesis of the extracted information
- the distribution of the results.

Name	Type	URL
ACM DL	Electronic Database	https://dl.acm.org/
IEEE Xplore	Electronic Database	https://ieeexplore.ieee.org/Xplore/home.jsp
Google Scholar	Electronic Database	https://dl.acm.org/

Table 1
List of databases explored in the search process

3. Conducting the SLR

3.1. Search and selection process

3.1.1. Database search

We carried out our search on three databases: IEEE Library, ACM Library, and Google Scholar. We considered these sources to be most suitable for our study due to their high accessibility and their technical content.

ACM DL is a database of full-text articles and bibliographic literature covering computing and information technology; IEEE delivers full-text access to technical literature, which contains information about engineering, computing, and technology information around the globe; and Google Scholar provides access to scholarly literature. A brief overview of them is provided in Table 1.

3.1.2. Merging, and duplicates and impurity removal

After conducting a systematic literature search, we identified 30 relevant papers related to black box testing. Upon further analysis of these papers, we identified several duplicate publications that appeared in multiple search results. We removed these duplicates to ensure that each paper was only counted once in our analysis.

We then screened the remaining 30 papers using pre-defined inclusion and exclusion criteria to select the most relevant papers for our analysis. After applying these criteria, we selected 10 papers for inclusion in our systematic literature review. Finally, we removed any inconsistencies in the data collected from the 10 selected papers by merging similar data points and removing any duplicate or overlapping data. This process ensured that our analysis was not biased and that we only reported accurate and reliable results.

In summary, the merging and duplicate removal process was an essential step in ensuring that our systematic literature review only included relevant and reliable sources of information. By removing duplicates and irrelevant information, we were able to ensure that our analysis was based on high-quality data and provided a comprehensive understanding of the existing literature on black box testing.

3.1.3. Application of the selection criteria

To further filter the studies found in the previous step, we define a set of criteria. A paper will be included if it satisfies the inclusion criteria detailed below:

I1. is full-text available,
I2. is peer reviewed,
I3. is written in English.
I4. contains "Black-box Testing" in the paper title or abstract,
I5. provides approaches for Black-Box Testing,
I6. provides an evaluation of the proposed solution.

The exclusion criterion is defined as follows:

E1. A study is excluded if it is a secondary or tertiary study (e.g., surveys, systematic literature reviews).

During the beginning of the study, we carried out two trials (containing 30 and respectively, 10 studies) with the reason to harmonize our understanding of the criteria and/or refine them, if needed. For both trails, it was essential that they fulfilled the following criteria: I1, I2 and I3. For the first trial, we focused more on the content from a broad point of view (criteria I3), while for the second trial we really browsed more through the papers, focused on criteria I5 and I6 and tried to keep only the more relevant and comprehensive ones. Applying the inclusion and exclusion criteria on the studies left from step 2, produced 10 research items. We mentioned that the filtering was manually achieved and we used the adaptive reading depth technique.

Id	Title	Author	Year
P1	Different Approaches To Black box Testing Technique For Finding Errors	Mohd Ehmer Khan	2011
P2	Black Box Testing based on Requirement Analysis and Design Specifications	Harsh Bhasin et al.	2014
P3	A comparative study on black-box testing with open source applications	Shaochun Xu, Lichao Chen, Chunning Wan, Oleksandr Rud	2016
P4	Combined testing approach: increased efficiency of black box testing	Zvonimir Kaprocki, Vukota Pekovic, Gordana Velikic	2017
P5	Black-Box Test Generation from Inferred Models	Petros Papadopoulos, Neil Walkinshaw	2015
P6	RESTest: automated black-box testing of RESTful web APIs	Alberto Martin-Lopez et al.	2021
P7	Comparison of Five Black-box Testing Methods for Object-Oriented Software	Kwang Ik Seo, Eun Man Choi	2006
P8	Tailoring of Black-Box Testing Methods	T. Murnane, K. Reed, R. Hall	2006
P9	Using Machine Learning to Refine Black-Box Test Specifications and Test Suites	Lionel C. Briand, Yvan Labiche, Zaheer Bawar	2008
P10	Antirandom testing: getting the most out of black-box testing	Y.K. Malaiya	2002

Table 2
List with selected articles

3.2. Data extraction

There is a total of 10 studies relevant to this work, all of them being enumerated in Table 2. These papers are identified by a unique ID, ranging from P1 to P10, and are listed in no particular order. The data extraction process was conducted manually using a data extraction form. The primary objective of the data extraction was to collect relevant information from the selected studies.

To extract data from the selected papers, we developed a data extraction form that included key fields such as author, title, year, research approach, evaluation metrics, and conclusions. The data extraction process was conducted manually to ensure the accuracy and completeness of the data. We carefully reviewed each study and extracted relevant information related to the research questions.

For *RQ1*, we focused on identifying different approaches and techniques used in black-box testing, such as requirement-based testing, or model-based testing. We also looked for information on the advantages and limitations of each approach. For *RQ2*, we searched for information related to the evaluation of black-box testing approaches. Specifically, we looked for keywords related to evaluation metrics, research results, and conclusions drawn from the evaluation process. Our goal was to identify the most commonly used evaluation metrics, as well as any limitations or challenges associated with the evaluation process.

3.3. Data synthesis

The data synthesis phase of the systematic literature review involved analyzing the data collected from the selected studies. We first organized the data into case studies based on the chosen topic and identified common themes and patterns across the studies. We then performed a qualitative analysis of the data, synthesizing the findings from the studies to draw meaningful conclusions. The synthesis included identifying similarities and differences between the approaches to black-box testing used in the selected works, evaluating the effectiveness of the approaches, and summarizing the evaluation results, as shown in Table 3.

Paper	Approach	Verification	Case study
P1	Equivalence Partitioning, BVA, Fuzzy Testing, OAT	The work describes several black-box testing approaches and how they work	academic
P2	Requirement-based Testing	The work involves creating a module description document, gathering input and output specifications for each module, determining ranges, and creating test cases	academic
P3	Equivalence Partitioning, BVA, Pairwise Testing, Decision Table Testing	The work compares four black-box testing techniques by testing them against two open source applications: ATC and YAPBAM	academic
P4	Hybrid approach: manual and automatic techniques	The work presents a hybrid testing methodology where manual and automatic black box testing are combined to increase the efficiency of BBT	industrial
P5	Model-Inference driven Testing (MINTEST)framework	The work provides a framework to generate better tests for components that lack source code	real life
P6	RESTest tool	The work provides an automated black-box testing tool for testing RESTful Web APIs that supports integration into continuous integration setups	real life
P7	Use-case driven testing, BBT using collaboration diagram, testing using extended use-cases, or formal specifications (OCL or Object-Z)	The work analyzes the behavior of 5 Black-Box Testing Techniques on systems developed using Object-Oriented Programming	academic
P8	Systematic Method Tailoring Procedures for BBT methods	The work describes how existing black box testing methods might be enhanced or created from scratch, based on the requirements elicitation phase	academic, industry
P9	Category Partitioning, Decision Trees	The paper presents MELBA, a C4.5 machine learning algorithm-based partially automated methodology to analyze and enhance the weaknesses and redundancies of test specifications and test suites	academic
P10	Generate anti random sequences for boolean inputs	The work presents a black-box approach that aims to maximise the test effectiveness by maintaining maximum diversity between tests	academic

Table 3
Data Synthesis Table

Different Approaches To Black box Testing Technique For Finding Errors:

Black box testing focuses on the functional requirement of the software. Selecting a black box test tool can be a challenging task due to the wide array of available commercial vendors and open-source projects in the area. We must take into consideration come factors such as: the ease of use, the cost, the accuracy, the test coverage, the test completeness, the reporting capability, the capacity of vulnerability database. The process of Black Box Testing consists of examining the input (requirement and functional specification of the system), the processing unit (test cases with selected input) and the output. After the desired output is obtained, a final report must be prepared. There are several techniques in Black Box Testing to finding errors. These are: Equivalence Partitioning, Boundary Value Analysis, Fuzzing, Cause Effect Graph, Orthogonal Array Testing, All-Pair Testing, State Transition Testing. The Equivalence Partitioning, as the name suggests, divides the input data of a software unit into partitions of data from which test cases can be derived. Its benefits are that it reduces the number of test cases. The Boundary value analysis focuses more on testing at boundaries or where the extreme boundary values are chosen. One of the important drawback of BVA is that it is only efficient for variable of fixed values. The Fuzzing technique is used for finding implementation bugs, memory leaks and security problems in software. Fuzzing can also suggest which part of program should get special attention, in the form of a code audit, application of static analysis, or partial rewrites. The test designs are simple to understand, the main limitation being that it generally only finds very simple faults. Another limitation with fuzzing is that whenever we perform any black box testing, we usually have a closed system to attack, which increases difficulty to evaluate the impact of the found vulnerability. Cause effect graph technique uses a graph and a relation between the effects and causes that are established. This technique is more difficult because the tester must identify the distinct causes and effect. Orthogonal array testing can be applied to problem in which the input domain is relatively small but too large to accommodate exhaustive testing. OAT is applied in system testing user interface testing, regression testing, configuration testing and performance testing. It reduces testing cycle time and provides uniformly distributed coverage of the test domain. With all the benefits there is one limitation with OAT that is; it does not guarantee the extensive coverage of the test domain. All pair testing uses test cases that are designed to execute all possible discrete combinations of each pair of input parameters. As no testing technique can find all bugs, pair wise testing is typically used together with other quality assurance techniques such as fuzz testing, unit testing and code review. State transition testing is designed to execute valid and invalid state transitions. It is useful when testing state machines and navigation of graphical user interface. It also tests various levels of coverage.

Black Box Testing based on Requirement Analysis and Design Specifications

The paper proposes a framework to prioritize the test cases based on requirement analysis and design. The purpose of requirement analysis is to refine the customer requirements based on performance, functions, and constraints. The approach addressed is to gather requirements from customers, sort them based on how feasible and achievable they are and then document them properly. As such, the activities performed during requirement analysis are requirement gathering, analysis and documentation. Software design, on the other hand, acts as a bridge between requirements analysis and development phase. All the gathered project requirements and objectives are transformed into project design, which is then used by the developer for coding phase. During this phase, the decision of what hardware and software will be used to build required product, what algorithms to use, what data structure, what flow and processes to follow is made. The objective of design documentation is to provide an efficient, modular design that will reduce the system complexity and result in an easy implementation. To test this approach, a comparison is made between Black Box and White Box testing. The White box testing as done by Cellular Automata. The generation of test cases by not considering the internal code has been accomplished using Artificial Life. It is desired to compare the test cases generated via Artificial Life and Lenten's loop with another technique. The system which generates overall test cases has two components. The first component generates test cases based on the code. The present work intense to replace the second component. The results obtained so far were encouraging. As a conclusion, requirement analysis is a crucial part of the software and the prioritization of test cases using requirement analysis and design specification is important to ensure the quality of the software.

A comparative study on black-box testing with open source applications

Software testing has become more complex as a result of software's ongoing growth in size and complexity. Although various testing techniques have been suggested, choosing the most effective technique remains the main challenge. By using the black box testing technique, a tester will interact with the system's user interface by providing inputs and evaluating results without any knowledge of the internal structure of the application. This paper aims to analyze and compare black-box testing techniques by conducting an experiment on two open source applications: Advanced Trigonometry Calculator and Personal Bank Account Manager. The techniques under evaluation are: Pairwise Testing (PWT), Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), and Decision Table testing (DTT). This research evaluates each technique's complexity to assess its efficiency, taking into consideration the following factors: number of test cases needed, numbers of detected bugs, the nature of discovered faults and time spent for testing. Results have shown that ECP approach detected the majority of defects in both applications and BVA revealed the least percentage of defects. However, a combination of different techniques may yield more precise results. Each approach plays a crucial role in the software testing process and the conducted experiment has proved that the exclusion of any of these approaches could potentially lead to a decrease in the number of identified bugs.

Combined testing approach: increased efficiency of black box testing

The complexity of software architecture has significantly increased in recent years which lead to higher demands of functional testing. Software testing relies significantly on manual testing, primarily because it has the ability to detect unique scenarios and edge cases. However, studies have shown that incorporating automatic and semi-automatic testing alongside manual testing can provide better cost benefit compared to relying on a single methodology. One important factor to mention is that automatic testing yields extensive coverage. This paper proposes an approach that can reduce the time and cost associated with the verification phase, while still ensuring the quality of testing, for the development, verification and manufacturing of Set-Top Box (STP) devices. A test based application, Test Management Application (TMA), is used to systematically track the testing process. This application manages the storage of testing requirements, test cases, results, and report generation of the results. The initial step of the testing process requires manual testing. We examine the order and number of test steps, as well as the test cases, to ensure that all potential fault scenarios that may emerge are covered. A similar procedure is performed during automatic testing. Automatic tests serve multiple purposes, including testing the performance of devices and conducting time-consuming long-term stress tests. Results have shown that the amount of time required for manual testing is impacted by the person performing the test, whereas in automatic testing, the duration of the test remains the same. In addition, an average time per test is longer for manual testing when compared to automatic testing. Manual and automatic testing both have a significant role in software testing. While manual testing reacts faster to unexpected changes and errors, automatic testing is considered more efficient and reliable.

Black-Box Test Generation from Inferred Models

Automated test generation typically requires either access to source code, or to some hand-crafted model of software behaviour. The downside is that models are rarely available , and they can quickly become outdated as the program evolves. In the absence of code or models, random testing is often used as an alternative. However, this approach can miss identifying faulty behavior that only arises from specific input combinations. This paper introduces a framework that aims to address the challenge of generating test inputs for components that lack source code ('black-box') and specifications. Specifically, the framework is designed to support the inference-driven test generation for non-sequential programs. The motivation for this paper was to develop an equivalent for LearnLib, which is an inference-testing toolset that has been particularly successful for sequential systems. However, this equivalent is designed for programs that do not necessarily operate on sequential inputs and outputs. The framework presented in this paper is called MINTEST. One noteworthy attribute of MINTEST is its deliberate modularity, which allows each component in the loop to be easily replaces by a variety of alternative implementations. This approach uses WEKA inference framework to infer suitable models from the resulting data by using the C4.5 decision tree inference algorithm. From one inferred model, Z3 solver is used to generate and execute tests from it. These tests are analysed to produce new test inputs, and the cycle iterates. To evaluate this framework, a preliminary experiment was conducted on three programs, using mutation testing as an approximation of test adequacy. Results demonstrated that the test sets generated by the framework are at least as adequate as equivalent random test sets, but they are more efficient.

RESTest: automated black-box testing of RESTful web APIs

RESTest is an automated black-box testing tool designed specifically for testing RESTful Web APIs. It is a framework that provides a comprehensive testing solution that covers all aspects of RESTful APIs, including functionality, performance, and security. RESTest takes in the OAS format specification of the API being tested and facilitates the creation and possible execution of test cases using advanced techniques such as adaptive random testing, constraint-based testing, and fuzzing. The input data required for the tests is automatically generated using custom test data generators. In this way, the test data required for each parameter (e-mail addresses, string matching a regular expression, currencies etc.) is realistically generated and further used in the test model. As far as obtained results are concerned, RESTest has already demonstrated its effectiveness in automatically identifying real-word bugs in widely-used commercial APIs. It is an open-source framework for automated black-box testing of RESTful Web APIs which incorporates specific testing strategies and test data generation techniques, enabling both online and offline testing, according to the needs of the user. Another primary benefit is that is supports integration into CI (continuous integration) setups, and thus it can be also used to monitor various RESTful services.

Comparison of Five Black-box Testing Methods for Object-Oriented Software

Black box testing is a method of software testing that involves testing a software application's functionality without any knowledge of its internal workings, architecture, or design. This testing technique is used at a system level as it executes a system by making use of the input data and the output results, without requiring any prior knowledge of the source code behind it. In this context, selecting a Black-Box Testing method is a process that requires great care and attention because the results of the test will vary according to the chosen technique. Similar comparisons of black-box testing techniques have been done in the past for other programming paradigms, such as the procedural paradigm. The aim of the current paper is to shift focus to the Object-Oriented paradigm by executing five different Black-Box testing methods based on two target systems developed by using Object-Oriented Programming: use-case driven testing, black-box testing using collaboration diagram, testing using extended use-cases, and testing using formal specifications(OCL or Object-Z). In Use Case Driven Testing, test cases are determined from basic use case diagrams, while the system algorithm and program module interaction are not taken into account. The second approach is based on creating a function-centered collaboration diagram from which test cases for invoking sequences of passing messages are extracted. Testing using Formal Specification consists of extracting test cases from Object-Z, a formal specification language for describing object-oriented system, or from building test cases from OCL by partitioning the domain of functions to be tested and expressing particular constraints in OCL. Lastly, testing using extended use cases consists of creating extended use cases from which scenarios are composed, classes and specific parameters are extracted and based on them, Method/Message paths are determined and further used in test case generation. The results of the experiments performed by comparing these 5 testing approaches focus on the coverage of each test case method over a domain divided in three components: System Exterior Domain, Modelling Domain, and Programming Domain. Extended Use Case method is the most effective method as it covers test cases in all parts of the domain, followed closely by Object-Z test method and Collaboration Diagram Test Method. Simple Use Case Test Method is only concerned with the System Exterior Domain, while OCL Test Method only tests the Programming Domain. The purpose of this empirical research is to show the variations in coverage for a couple of different black box testing approaches, highlighting the importance of the development of high quality software where user requirements and specifications are properly states prior to system development.

Tailoring of Black-Box Testing Methods

Although the effectiveness of black-box testing methods has been proved time and time again, most of these methods are still incomplete and often require in-place customization for certain test-cases. The aim of the paper is to present methodologies for black-box testing methods in order to further facilitate their development. The approach with respect to this is by applying GQAS (Goal/ Question/ Answer/ Specify), a requirement elicitation technique that helps identifying information required in the black-box testing process, such as the datatype, set type and size in terms of minimum and maximum lengths of input/ output data. Three Systematic Method Tailoring procedures are then used in creating optimal black-box testing methods or tailoring the already-existing ones. The first one is selection-based tailoring which is an approach based on existing black-box techniques that enables the creation of new methods by matching the rule types in the generalized representation against the valid data types for each field that is being tested. Creation-based tailoring is focused on generating new rules for existing methods and is particularly useful when testers have reason to believe that a specific input may be effective for testing a particular field. The third method tailoring procedure, Creation-Based Tailoring via Selection, is a mix between the first two and enables the creation of new rules based on existing ones. The procedures illustrated above are evaluated by being applicated to an internet-based application and the results suggested that they can be easily incorporated into academia and industry-related testing as they facilitate the incorporation of testers into the requirements elicitation process and are an opportunity to further explore the field of research in terms of Black Box Testing

Using Machine Learning to Refine Black-Box Test Specifications and Test Suites

This paper proposes a partially automated methodology to help software engineers analyse the weaknesses of test suites and iteratively improve them. The chosen approach is based on abstracting test suite information and then deciding about changes to the test suite. To transform test cases into test case specifications at a higher level of abstraction, they rely on Category-Partition, a black-box test specification technique. Test cases are abstracted under the form of category and choice combinations, as defined in Category Partition. These choice combinations characterize a test case in terms of input and execution environment properties. A machine-learning algorithm is then used to learn about relationships between inputs/environment conditions and outputs as they are exercised by the test suite. This allows the tester to precisely understand the capabilities and weaknesses of the test suite. Based on a series of systematic heuristics to guide the analysis of those relationships, this approach then facilitates the improvement of the test suite specification and test cases. The motivation is to ensure that the behaviour of the software under test is fully exercised. Once the categories and the choices are defined, they are then used to automatically transform test cases into "abstract" test cases. Devising such categories and choices is necessary to understand the rationale behind test cases and is a way for the tester to formalize the understanding of the functional specifications of the software under test. The main reason to transform the test suite into an abstract test suite is that it will be much easier, for the machine learning algorithm to learn relationships between input properties and output equivalence classes. First, problems must be identifed in C4.5 Trees and then they must be linked to causes. Then, if needed, the cause is added to a test case. Examples of such reasonings could be a choice or category is missing in the tree or certain choice combinations are missing. There are two solutions for when you are asking which combination with other choices to include in the test suite. The first solution is to follow the CP method and build all the feasible (according to properties and selectors) combinations of choices and select the ones that are missing in the abstract test suite. An alternative is to identify which combinations of choices may be relevant to determine the output class and could be missing from the test suite. The case study took place in the context of a specialized 4th year course on software testing. 21 students were properly trained regarding white and blackbox testing techniques, including CP. They were asked, during a three-hour lab period, to devise a test specification from the source code using CP, and devise a test suite from this specification. As a conclusion, using MELBA they were able to identify instances of problems in the decision trees and use this information to improve both test suites and CP specifications. Another conclusion they made was that from the case study, the taxonomies of decision tree problems and their possible root causes are complete with respect to the PackHexChar program.

Antirandom testing: getting the most out of black-box testing

Unlike random testing (which randomly selects a test to run regardless of the previous selection), antirandom selects tests such that its total distance from all previous tests is maximum. As such, the selection of each test explicitly depends on the tests already obtained. In general, the input variables for a program can be numbers, characters, and data structures. For generating antirandom sequences for boolean inputs. to make each new test as different as possible, Hamming distance and Cartesian distance must be used as measures of difference. An approach chosen is that using the hypothesis that if two input vectors have only a small distance between them then the sets of faults encountered by the two is likely to have several faults in common. Conversely, if the distance between two vectors is large, then the set of faults detected by one is likely to contain only a few of the faults detected by the other. There are three main procedures that can be followed: Construction of a MHDATS, Expansion of MHDATs, Expansion and Unfolding of MHDATs. The obtained results based on the construction of MHDATSs and MCDATSs are that if a sequence B is obtained by reordering the variables of sequence A, then B is a variable order-variant (VOV) of A and if a binary sequence B is obtained by changing the polarity of one or more variables of a sequence A, then B is termed a polarity-variant of A. The observations based on the expansion of MHDATS were that it is always possible to extend a MHDATS (MCDATS) by adding one more variable this procedure. a formal proof for this is being sought, the column added is a function of columns already included. Another approach based on domain and partition analysis and the concepts of equivalence partitioning, revealing subdomains and homogeneous subdomain is discussed. The technique partially encodes an input into binary, such that sample points desired can be obtained by automatic translation. In conclusion, this black-box approach attempts to maximize the test effectiveness by keeping tests as different as possible from each other

4. Results

4.1. RQ1 - What approaches have been developed to support Black-Box Testing?

Several approaches have been developed to support black box testing in software development. Some of the most commonly used approaches are:

1. Equivalence Partitioning: As the name suggests, divides the input data of a software unit into partitions of data from which test cases can be derived. Its benefits are that it reduces the number of test cases.
2. Boundary Value Analysis: Focuses more on testing at boundaries or where the extreme boundary values are chosen. One of the important drawback of BVA is that it is only efficient for variable of fixed values.

3. Fuzzing: used for finding implementation bugs, memory leaks and security problems in software. Fuzzing can also suggest which part of program should get special attention, in the form of a code audit, application of static analysis, or partial rewrites.
4. Cause Effect Graph: Uses a graph and a relation between the effects and causes that are established. This technique is more difficult because the tester must identify the distinct causes and effect.
5. Orthogonal Array Testing: Can be applied to problem in which the input domain is relatively small but too large to accommodate exhaustive testing. OAT is applied in system testing user interface testing, regression testing, configuration testing and performance testing. However, it does not guarantee the extensive coverage of the test domain.
6. All-Pair Testing: Uses test cases that are designed to execute all possible discrete combinations of each pair of input parameters. As no testing technique can find all bugs, pair wise testing is typically used together with other quality assurance techniques such as fuzz testing, unit testing and code review.
7. State Transition Testing: Designed to execute valid and invalid state transitions. It is useful when testing state machines and navigation of graphical user interface. It also tests various levels of coverage.
8. Decision Table Testing: This approach involves creating a table that lists all possible combinations of inputs and expected outputs. Test cases are designed to cover all possible combinations to ensure that the software is functioning correctly.

4.2. RQ2 - How have Black-Box Testing approaches been evaluated and what were the reached conclusions?

Black-box testing techniques have been evaluated and compared in various studies. In one study, four techniques were evaluated: Pairwise Testing (PWT), Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), and Decision Table testing (DTT), on two open-source applications. The ECP approach was found to be the most effective in detecting the majority of defects, while BVA revealed the least percentage of defects. However, a combination of different techniques may yield more precise results. Each approach plays a crucial role in the software testing process, and the exclusion of any of these approaches could potentially lead to a decrease in the number of identified bugs.

In another study, manual testing was compared with automatic testing for the development, verification, and manufacturing of Set-Top Box (STP) devices. The study found that both manual and automatic testing have a significant role in software testing. While manual testing reacts faster to unexpected changes and errors, automatic testing is considered more efficient and reliable. The initial step of the testing process requires manual testing to ensure that all potential fault scenarios that may emerge are covered, while automatic tests serve multiple purposes, including testing the performance of devices and conducting time-consuming long-term stress tests.

A framework called MINTEST was introduced to address the challenge of generating test inputs for components that lack source code ('black-box') and specifications. This framework is designed to support the inference-driven test generation for non-sequential programs, using the C4.5 decision tree inference algorithm. To evaluate this framework, a preliminary experiment was conducted on three programs, using mutation testing as an approximation of test adequacy. Results demonstrated that the test sets generated by the framework are at least as adequate as equivalent random test sets, but they are more efficient.

Lastly, another approach is based on GQAS (Goal/ Question/ Answer/ Specify), a requirement elicitation technique that helps in identifying information required in the black-box testing process, such as the datatype, set type and size in terms of minimum and maximum lengths of input/ output data. Three Systematic Method Tailoring procedures are then used in creating optimal black-box testing methods or tailoring the already-existing ones. These include selection-based tailoring, creation-based tailoring, and creation-based tailoring via selection.