JOURNAL LA MULTIAPP

VOL. 03, ISSUE 03 (104-112), 2022
DOI: 10.37899/journallamultiapp.v3i3.648

Review of Software Testing Methods

Yasmin Makki Mohialden[1], Nadia Mahmood Hussien[1], Sundos Abdulameer Hameed[1]

[1]*Computer Science Department, Collage of Science, Mustansiriyah University, Baghdad, Iraq*

**Corresponding Author: Yasmin Makki Mohialden*
Email: ymmiraq2009@uomustansiriyah.edu.iq

Article Info

Article history:
Received 5 May 2022
Received in revised form 14 June 2022
Accepted 18 June 2022

Keywords:
Test Driven Development,
Software,
Automation Testing,
Test Optimization,
S Software Testing,
Black Box Testing,
White-Box Testing

Abstract

With the increasing complexity of programs comes an increased focus on ensuring the quality of these programs. Essentially, it depends on improving the methods of testing the quality of these programs in the two phases of building these programs and after their operation. Therefore, we are developing software testing methods and methodologies. This paper aims to discuss software testing methods and their classifications according to their main properties and any software that suits each method. The majority of the literature on software testing methods and techniques is also included.

Introduction

Testing is the process of assessing whether or not a system satisfies the criteria that were first outlined for it. The process of verifying that the produced system satisfies the criteria given by the user is referred to as validation. In this specific situation, the actual results and the expected results are different from one another. Throughout the process of testing software, deficiencies, errors, and prerequisites are found in the application or system that has been produced. Inquiring about the quality of the product is one way to gather information that is particular to the product.

Testing software is exactly the same as any other endeavour that involves taking risks. In order to accomplish this goal, testers need to be capable of handling large numbers of tests while also making decisions on whether or not risks need further investigation. One of the goals of efficient testing is to complete the fewest feasible tests in order to get as much information as possible. Testing software is an essential part of both ensuring the quality of software and controlling its quality. Due to the possibility of schedule delays, cost overruns, and cancellations, software testing for mission-critical applications might end up being rather expensive (Jamil et al., 2016).

The process of testing is divided up into a number of different levels and stages, and the tester will go from one level to the next as they proceed through the process. System testing may be broken down into three primary stages: testing at the unit level, testing at the integration level, and testing at the system level.

A software developer and a software tester will each give each of these stages their full attention before considering any of them to have reached their respective completion points. Testing the

program is one of the many stages that must be taken in order to develop a dependable software system. There are two distinct categories, which are referred to respectively as static and dynamic sorts.

The practice of visually reviewing program code and accompanying documentation is referred to as static testing. This kind of testing does not include actually running the program being tested. The usage of symbolic execution is a common approach that is used while carrying out the process of static testing.

The purpose of dynamic testing, on the other hand, is to validate the software by seeing how it reacts when subjected to a variety of inputs. This is done to guarantee that the program is operating as intended (Korel, 1990; Guo et al., 2022). The software development lifecycle includes iterative testing as one of its components (SDLC). In order to successfully build software, it is necessary to organize separate teams for each module (unit testing).

Software architectures that only include a single module are called single-module systems. After integration, it is not unusual for there to be errors. System testing, which encompasses evaluating the program in its whole, is the last test in the software development life cycle (SDLC). The programming procedure is not hindered in any way by the integrated units' absence of complications. It is possible for testing complex systems on a large scale or in depth to require a substantial amount of both time and resources. Having more components than necessary makes it more difficult to test each possible combination and state of affairs, which is another another downside of having more components than necessary.

The practice of testing may be used to many different kinds of development activities across many different types of businesses. In spite of the fact that there is a vast variety of testing scenarios and applications, there is still a relationship between the goals. All initiatives pertaining to the expansion of the company are put through exhaustive testing to guarantee that they will not wind up costing more than was originally estimated or falling through entirely.

This primary motivator presents four exam objectives that are associated with one another. Determine the extent of the development risks and the causes of those risks that may be reduced by testing and experimentation. The identified risk might potentially be mitigated by the use of testing. When will the last phase of testing take place? Check to see that the testing is under control throughout the whole of the development process (Dwivedi et al., 2022; Everett & McLeod, 2007).

The remaining parts of the paper have the following organizational format: The previous research on software testing is analyzed and discussed in Section 2 of this study. Techniques for testing software are covered in Section three, and the conclusion is presented in Section four.

A Review of the Literature on Software Testing

Jovanovi (2006) provides a concise overview of the most important testing techniques and methods. The concepts of black box testing and white box testing are presented, together with the accompanying testing procedures, in the context of a discussion on generic classification. The following are some examples of these: (Comparative Testing, Equivalent Partitioning, Boundary Value Analysis, and Graphing of Cause and Effect)

Roshan et al. (2012) conducted an analysis of the many search-based software testing approaches, made projections on future research trends, and took a peek into the future of software testing. This paper includes an in-depth summary of Search-Based Software Testing (SBST) as well as other contemporary computing fields that overlap with SBST.

Within the scope of this study, we investigate recent advancements in search-based software testing. In the context of contemporary software testing, the area establishes a new field.

Search-based software testing is more efficient than other techniques of software testing, which results in time and financial savings. The search-based method of software testing is adaptable enough to be used in either a white-box or a black-box setting.

Other methods of contemporary computing, such as genetic algorithms and finite state machines, may be implemented using it (FSM). It is clear that the scientific world is becoming more interested in the very promising branch of software testing that is Search-Based Software Testing as seen by the rising number of articles devoted to this topic.

Sawant et al. (2012) examines the value of doing software quality assurance testing. The goals and fundamental tenets of software testing are also scrutinized in this study. They discuss a variety of testing procedures and approaches for different types of software. The last benefit is that it makes the line between software testing and debugging more clearer.

According to Gautam et al. (2022), software testing automation is an approach that can be successfully used in software engineering. On the other hand, they discovered 48 main research papers for each different machine learning-based software testing approach. This demonstrates how the model uses the data to inform its learning and prediction processes. They made the discovery that the most common use of machine learning was for the purpose of developing, improving, and evaluating test cases. In addition to that, the expenses of using the Oracle database have been estimated using machine learning.

The subject of object-oriented software testing is investigated in Raza et al (2022)'s study. Object-oriented software testing makes use of a variety of methods and methodologies, some of which include UML diagrams, evolutionary testing, genetic algorithms, black-box testing, and white-box testing. In order to figure out what was going on, our research conducted a review of the most current relevant literature.

Umar and Zhanfang, in their research from 2020, examine and compared a number of different dynamic software testing approaches. In this post, we will talk about some of the most common testing approaches that are utilized in situations that are constantly changing. A comparison of different dynamic testing approaches is carried out as the last step in an effort to enhance testing and, as a result, the quality of software. On the other hand, this article analyzes and contrasts the dynamic software testing methodologies that are used most often. In software development, the use of these dynamic software testing methodologies may assist testers perform a good job of testing, which in turn improves the quality and reliability of the program.

A literature study on the application of test levels, test sorts, test design methodologies, and test automation/tools was carried out by Backstrom (2022). With the exception of unit testing, the findings vary quite a bit depending on the demography that was researched. In descending order of popularity, the most common tests are functional ones. Testing for security is performed less often than testing for regression, performance, and usability respectively.

Methods that are considered to be black boxes, such as use case testing and boundary value analysis, as well as exploratory testing and error guessing, are all helpful approaches to get an understanding of how things operate. White-box approaches are not nearly as widespread as their black-box counterparts. The fact that just a small proportion of tests are often automated, on the other hand, indicates that total utilization is not very high. Testing that is performed by hand plays a vital role in the business.

Ahuja et al. (2022) present a summary of several software testing approaches, such as differential, metamorphic, mutation, and combinatorial testing, as well as adversarial perturbation testing, and discuss some challenges in their deployment for augmenting perception systems that are used in VBS. Specifically, the authors focus on the application of these testing methods.

In addition to this, we provide and discuss the findings of a preliminary experimental comparison to an established benchmark that is used in VBS. In this study, they discussed several testing methodologies for DL models. For instance, we discovered that integrating testing approaches is an effective way to handle problems like as the quality of models and training datasets, the identification of oracles, and the selection of test inputs.

In spite of the fact that DL models have their own unique issues, we have come to the realization that conventional approaches to software testing may be combined to test these models. Aspects such as differential testing, metamorphic testing, and adversarial perturbation testing are desired because they reduce the test oracle issue and detect adversarial assaults.

Other examples of testing include metamorphic testing and adversarial perturbation testing. The fault-revealing metamorphic relations are not categorized, but the pictures and objects themselves are. The purpose of mutation testing is to determine whether or not there is variation in the output of a model when given inputs that are otherwise identical.

In spite of the challenges involved in designing mutation operators, the possibility of using them to test DL models is quite interesting. You won't have to manually create new test cases thanks to this feature, so you can analyze the model as well as the datasets. DL models may be placed through adversarial perturbation tests, which is an intriguing feature of these models since it enables them to be evaluated for their resistance to change as well as their level of safety.

On the basis of the author's research and experience, Wang and Li (2022) provide a summary of testing strategies that are appropriate for large-scale information software. Article: This study may be useful in the testing of large-scale information software, as well as the author's own work, which may be improved as a result.

The Techniques of Software Testing

This section provides an overview of the many software testing approaches that are currently available. The following are the two categories that are used to classify methods to software testing:

Manual testing, often known as static testing: The process of testing is one that is drawn out and laborious. In terms of statistics, it is carried out at an earlier stage in the life cycle. In certain circles, it is also referred to as "static testing." Analysts, developers, and testers are the ones who carry it out. The following are just a few of the various manual testing approaches that are available: A look at the information, a tour of the facility, a study of the technical aspects, and an examination (Sawant et al., 2012).

Testing that is dynamic (also known as automated testing): This tester does the testing by executing the script on the testing tool and carrying out the procedures. There are a few different types of automated testing, and one of them is called dynamic testing. Additional types of automated testing include the following categories and subtypes: Testing to determine whether or not it is accurate, how quickly it can be done, how safe it is, and how long it can keep going. Testing for correctness, performance, security, and reliability are the names given to these different kinds (Sawant et al., 2012).

Testing for correctness requires that software at least fulfill the fundamental level of accuracy expectations. When testing for correctness, it will be necessary to consult an oracle in order to differentiate between permissible and undesirable forms of behavior. The tester could have previous knowledge of the inner workings of the software module that is now being tested, although this is not guaranteed. As a consequence of this, testing strategies such as black-box, white-box, or grey-box might be used (Khan, 2010).

White box testing examines not just how well the program performs its intended functions but also how well it adheres to its design specifications. Testing in a white box environment demands a strong understanding of programming. White box tests are also often referred to as "transparent boxes" or "glass boxes" testing. This methodology may be used for testing at the unit, integration, and system levels. In other words, it establishes whether or not data is protected by information systems and whether or not they operate effectively. As a result, it is able to test the individual pathways that a module may take. All loops and data structures are put through their paces in the testing process.

White box testing is a complex testing method that also requires programming. It is conducted by testing professionals. It does a thorough inspection and examination of the code by examining the module. In the event that the output does not conform to the prerequisites, it will be recompiled and examined again. The pure code of an application is evaluated, not its user interface or its visual presentation.

This kind of testing examines how a software is put together on the inside. It involves providing input to a system and seeing it while it processes that data in order to create an output. A tester has to be familiar with the source code. Testing in a white-box environment is essential for all other types of testing, including integration, unit, and system testing. It guarantees that all test objects and components are executed in the correct manner (Bourque et al., 2002; Andriyadi et al., 2022; Khan, 2010; De Troyer & Leune, 1998).

The testing of the black box: It is based on an analysis of the specifications of a piece of software rather than taking into account how the program really works on the inside. The purpose of this test is to determine how effectively the component satisfies the criteria that have been established for the component.

It gives the system's underlying logical structure either very little or no attention at all, instead concentrating entirely on the system's most important characteristics. It ensures that the input is successfully received and that the output is generated correctly. The integrity of the external data is preserved during the black box testing.

"Black box" testing may refer to a variety of different types of testing, including exploratory testing, functional testing, stress testing, load testing, exploratory testing, usability testing, smoke testing, recovery testing, and volume testing.

Tests and evaluations (alpha and beta) Some of the alternatives to traditional black box testing include graph-based testing and equivalence partitioning, boundary value analysis and comparison testing, orthogonal array and comparison testing, specialized testing, fuzz testing, and traceability metrics (Sawant et al., 2012; Jiang & Hassan, 2015).

Testing Using Grey Boxes Grey box testing is a hybrid approach that involves both white box and black box testing. The purpose of grey box testing is to ensure that software complies with its specifications while also providing insight into its inner workings. This involves using reverse engineering to discover error messages or boundary values.

The method of evaluating software while being aware of the code or logic that lies under the surface is known as grey box testing. When compared to black-box testing, grey-box testing calls for a more comprehensive knowledge of the program's inner workings. Tests conducted in a black box or gray box do not provide examination of all of the system's internal components (Sachtleben & Peleska, 2022).

Performance Testing

This kind of testing is used to assess whether or not a system or component satisfies a certain requirement for its level of performance. This comprises the amount of resources used, the amount of work done, and the amount of time it takes to react to stimuli. Performance testing

is a method that may be used to assess an application's level of functionality. It works hard to keep a website's latency, throughput, and utilization rate as low as it can while still meeting its goals. It is a single field of study that encompasses all aspects of business operations, including planning, design, implementation, analysis, and reporting. It does not consist of any individual components. there are two different kinds of performance tests:

Testing under load: It is necessary for several huge software systems to execute hundreds of thousands or perhaps millions of inquiries concurrently. These systems need extensive testing to guarantee that they will function at their highest potential (i.e., how quickly requests come in). The maximum capacity of a computer, peripheral, server, or network, as well as an application, may be determined via a procedure known as load testing.

You may use load testing to analyze the capabilities of a single system in great detail, or you can use it to compare and contrast the capabilities of numerous systems. By analyzing the results of this test, one may evaluate whether or not the program is able to handle a high number of users (Sawant et al., 2012; Jiang & Hassan, 2015).

The term "stress testing" refers to a kind of testing in which random processes are carried out at a larger volume, speed, and for a longer period of time than is typical. It is an approach that will speed up the process of discovering problems and assessing the product's durability. Stress testing is one kind of testing that may be done. It is the process of testing a system or component beyond the limitations that have been claimed for it in order to determine how it fails. More stress testing is required as the user base continues to increase.

A huge number of inputs, questions, and other types of information are obtained during stress testing (Khan, 2010). It is used to establish how a system or component breaks down when subjected to strain (Khan, 2010). Consider doing a stress test to see how the responsiveness of the user interface shifts in response to changes in the CPU, RAM, storage, and network workloads (Fu et al., 2022).

Testing for security purposes guarantees that only people with appropriate permissions may access the software. Any developed system or one that is still in the development stage is susceptible to suffering severe damage if key security flaws and vulnerabilities are exploited. Testing for security helps find problems and find solutions to those problems. It assures that the system will continue to function normally over the long term. Additionally, it protects systems against intrusions that were not allowed. Testing for security vulnerabilities is beneficial to the organization as a whole (Khan, 2010). There is a direct correlation between quality, reliability, and safety. Software vulnerabilities make it possible for hackers to bypass security.

The program and its features are only accessible to people who have been given permission to use them. Firewalls are used to detect data leakage. This is accomplished by encrypting the application and using a variety of software and hardware (Sawant et al., 2012). Testing for software security aims to uncover software faults and vulnerabilities in an effort to prevent the loss of data (Haridas, 2007). Finding software's flaws and potential dangers is an important step in protecting it from being misused.

It is possible to either make the software function or make it not operate if it is exploited in an effective manner. The purpose of software security testing might vary greatly from one organization to the next (Arkin et al., 2005; Yimer & Gizachew, 2022). The security testing methods that are used the most often are expensive and, at times, excessively complicated in the activities and phases that they include. Because of the difficulty, many developers have a tendency to disregard software testing for security (Yimer & Gizachew, 2022).

Testing Reliability Early on in the design phase, reliability testing is carried out in order to ensure that the system satisfies the reliability criteria that were established for it.

The testing of reliability involves a lot of interconnected parts. This testing uses software that already has a mechanism for testing built into it. It is an effective method for determining the reliability of software. The reliability of a system is determined after the software has been developed. Methods of either analyzing or fixing may be used in the testing of the software (Sawant et al., 2012). Testing, estimating, and predicting the reliability of a system also demonstrates how to create test plans that provide data on failure and/or deterioration in a very short amount of time.

A comparison of test plans that have reliability estimates that are comparable is also included in this discussion. This article discusses the use of degradation data for the purpose of reliability estimate and maintenance decision making (Elsayed, 2012).

Conclusion

Even while the infrastructure for the creation and testing of software is always being improved, its importance to the process cannot be overstated. Testing, on the other hand, is often performed as the very last stage in the process of developing software. This research presents the most essential prior works linked to the study of software testing techniques as well as the most important ways for testing existing software based on current sources. Additionally, the most important methods for testing new software are also discussed. Each procedure was broken down into its component parts, along with the parameters for how it should be carried out and the allotted amount of time for it to be completed. testing until later in the system's life cycle, and not only until the late stages, because the system's testing process occurs late in the life cycle of all software systems, and not only in the late stages. system's financial resources and lost time by delaying testing until later in the system's life cycle, and not only until the late stages.

Acknowledgement

the Authors would like to thank Mustansiriyah University (https://uomustansiriyah.edu.iq/) Baghdad -Iraq for its support in the present work.

References

Ahuja, M. K., Gotlieb, A., & Spieker, H. (2022, April). Testing Deep Learning Models: A First Comparative Study of Multiple Testing Techniques. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 130-137). IEEE.

Andriyadi, A., Fikri, R. R. N., & Saputri, E. F. (2022). Evaluasi Sistem Informasi Perpustakaan Institut Informatika Darmajaya Dengan Whitebox Testing. *Journal of Innovation Research and Knowledge*, *1*(8), 743-746.

Arkin, B., Stender, S., & McGraw, G. (2005). Software penetration testing. *IEEE Security & Privacy*, *3*(1), 84-87.

Bäckström, K. (2022). *Industrial Surveys on Software Testing Practices: A Literature Review* [Thesis]. University of Helsinki.

Bourque, P., Lavoie, J. M., Lee, A., Trudel, S., & Lethbridge, T. C. (2002, October). Guide to the software engineering body of knowledge (swebok) and the software engineering education knowledge (seek)-a preliminary mapping. In *Proceedings 10th International Workshop on Software Technology and Engineering Practice* (pp. 8-8). IEEE Computer Society.

De Troyer, O. M. F., & Leune, C. J. (1998). WSDM: a user centered design method for Web sites. *Computer Networks and ISDN systems*, *30*(1-7), 85-94.

Dwivedi, N., Katiyar, D., & Goel, G. (2022). A Comparative Study of Various Software Development Life Cycle (SDLC) Models. *International Journal of Research in Engineering, Science and Management*, *5*(3), 141-144.

Elsayed, E. A. (2012). Overview of reliability testing. *IEEE Transactions on Reliability*, *61*(2), 282-291.

Everett, G. D., & McLeod, R. (2007). *The Software Development Life Cycle*. Wiley-IEEE Press. https://doi.org/10.1002/9780470146354.ch2

Fu, J., Wang, Y., Zhou, Y., & Wang, X. (2022). How resource utilization influences UI responsiveness of Android software. *Information and Software Technology*, *141*, 106728.

Gautam, S., Khunteta, A., & Sharma, P. (2022). A Review on Software Testing Using Machine Learning Techniques. *ECS Transactions*, *107*(1), 3393.

Guo, X., Okamura, H., & Dohi, T. (2022). Automated Software Test Data Generation With Generative Adversarial Networks. *IEEE Access*, *10*, 20690-20700. https://doi.org/10.1109/ACCESS.2022.3153347

Haridas, N. (2007). Software Engineering-Security as a Process in the SDLC. *SANS Institute*, 29.

Jamil, M. A., Arif, M., Abubakar, N. S. A., & Ahmad, A. (2016, November). Software testing techniques: A literature review. In *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)* (pp. 177-182). IEEE. https://doi.org/10.1109/ICT4M.2016.045

Jiang, Z. M., & Hassan, A. E. (2015). A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, *41*(11), 1091-1118.

Jovanović, I. (2006). Software testing methods and techniques. *The IPSI BgD Transactions on Internet Research*, *5*(1) 30-41.

Khan, M. E. (2010). Different forms of software testing techniques for finding errors. *International Journal of Computer Science Issues (IJCSI)*, *7*(3), 24.

Korel, B. (1990). Automated software test data generation. *IEEE Transactions on software engineering*, *16*(8), 870-879. https://doi.org/10.1109/32.57624

Raza, A., Shah, B., Ashraf, M., & Ilyas, M. (2022). Object-Oriented Software Testing: A Review. In *Proceedings of International Conference on Information Technology and Applications* (pp. 461-467). Springer, Singapore.

Roshan, R., Porwal, R., & Sharma, C. M. (2012). Review of search based techniques in software testing. *International Journal of Computer Applications*, *51*(6).

Sachtleben, R., & Peleska, J. (2022). Effective grey-box testing with partial FSM models. *Software Testing, Verification and Reliability*, *32*(2), e1806.

Sawant, A. A., Bari, P. H., & Chawan, P. M. (2012). Software testing techniques and strategies. *International Journal of Engineering Research and Applications (IJERA)*, *2*(3), 980-986.

Umar, M. A., & Zhanfang, C. (2020). A Comparative Study of Dynamic Software Testing Techniques. *International Journal of Advanced Networking and Applications*, *12*(3), 4575-4584.

Wang, B., & Li, H. (2022). Analysis of Testing Methods of Large-scale Information Software. *Academic Journal of Computing & Information Science*, *5*(1).

Yimer, S., & Gizachew, B. (2022). The Development of a Web-Based Application Security Testing Framework in Addis Ababa, Ethiopia. *Global Journal of Computer Sciences: Theory and Research, 12*(1), 12-22.

Yimer, S., & Gizachew, B. (2022). The Development of a Web-Based Application Security Testing Framework in Addis Ababa, Ethiopia. *Global Journal of Computer Sciences: Theory and Research, 12*(1), 12-22.