TSE 3251

SOFTWARE VERIFICATION AND VALIDATION

Functional and Non-Functional Testing Techniques

Ahmed Hassan Mohammed Salem

(1191102340)

TRIMESTER 2, 2022/2023

FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

27th JUNE 2023

1. Introduction

1.1 Functional & Non-Functional Testing Techniques

In the field of software development, ensuring the quality and reliability of software systems is of great importance. Testing is essential to this process since it systematically assesses software products to find flaws and ensure that they adhere to specifications. Software testing is divided into two techniques: Functional testing, and non-functional testing.

Functional testing focuses on validating the functional behavior of a software system. These types of testing techniques confirm that a software system functions as intended. It seeks to ensure that the system correctly carries out its intended functions and generates the anticipated outputs. This sort of testing focuses primarily on its functional requirements. Various testing techniques, including unit testing, integration testing, system testing, and acceptance testing, are frequently used in functional testing.

While functional testing focuses on the system's behavior, non-functional testing concentrates on evaluating the quality attributes of a software system. Non-functional testing looks at elements such as performance, scalability, stability, usability, security, and compatibility that go beyond the basic capabilities. Techniques involved in Non-Functional testing includes performance testing, usability testing, security testing, reliability testing, and compatibility testing.

1.2 Objectives

The objectives of Function Testing techniques are as follows:

- To ensure that the software system performs the intended functions accurately and reliably.
- To ensure that the software system produces the expected outputs based on specified inputs and conditions.
- To ensure that the software system adheres to the functional requirements defined in the software specifications.

The objectives of Non-Functional Testing techniques are as follows:

- To evaluate and validate the software system's performance and responsiveness under different workload conditions.
- To evaluate and validate the software system's reliability and stability in terms of system availability and error handling.
- To evaluate and validate the software system's security and protection against unauthorized access, data breaches, or vulnerabilities.

1.3 Existing Functional Testing methods

1.3.1 Unit Testing

Individual components or modules of the software system are tested individually during unit testing. It checks that each component operates properly and generates the desired results in accordance with predetermined inputs and conditions. Prior to being incorporated into the larger system, this technique ensures the reliability of individual units and assists in the early detection of flaws.

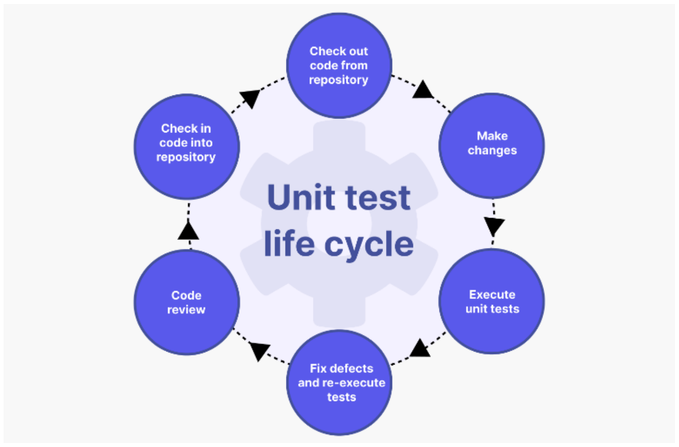

Unit test life cycle
Check out code from repository
Make changes
Execute unit tests
Fix defects and re-execute tests
Code review
Check in code into repository

Figure 1.1 shows the life cycle of Unit Testing (Katalon-Unit testing, n.d.)

1.3.2 Integration testing

Integration testing verifies the communication and cooperation between various software system modules or components. It confirms that the integrated components operate as expected, sharing information, and effectively carrying out functionalities. Integration testing aids in the discovery of errors resulting from mismatched interfaces, inconsistent data, or problems with interoperability.

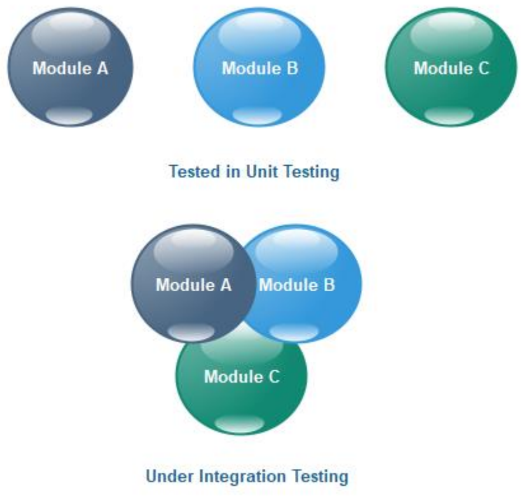


Figure 1.2 shows the methodology of Integration testing process (Javatpoint-Integration Testing, n.d.)

1.3.3 System Testing

In order to ensure that the software system as a whole complies with the functional requirements, system testing is performed. From beginning to end, it investigates the system's operation and behavior while addressing a range of use cases and situations. System testing guarantees that the system performs as planned, conforms to user expectations, and appropriately handles various inputs and conditions.

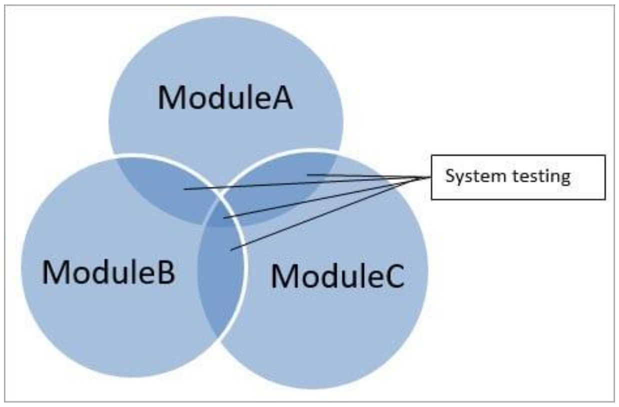
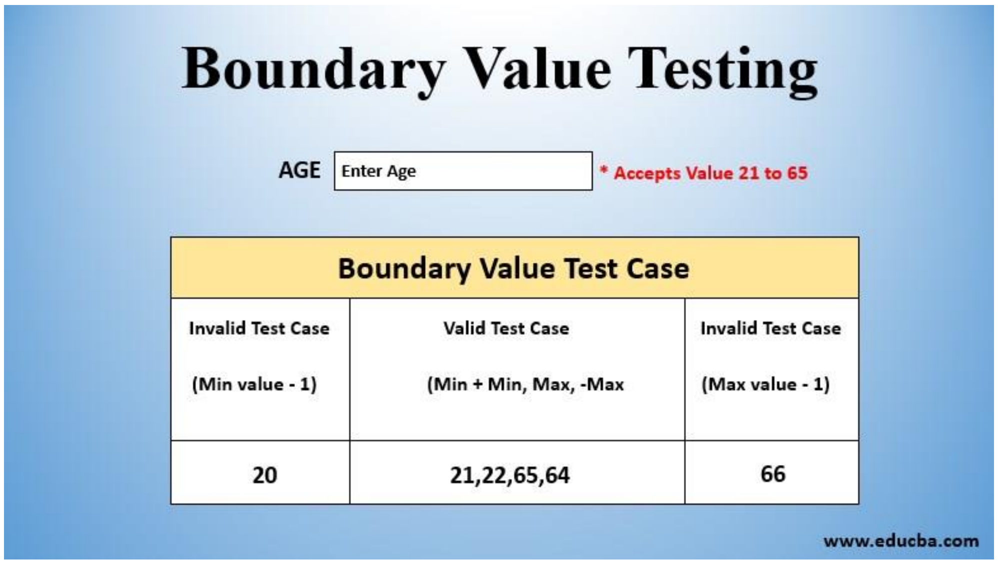


Figure 1.3 shows the methodology of the System testing process.

(softwaretestinghelp-system-testing, n.d.)

1.3.4 Boundary Value Testing

Testing the limits or boundaries of input values and conditions is the focus of boundary testing. It attempts to confirm that inputs at or close to the prescribed boundaries are correctly handled by the software system. Boundary testing aids in locating problems with data validation, boundary conditions, and edge cases by testing both the lower and upper borders.


Boundary Value Testing

AGE Enter Age * Accepts Value 21 to 65

Boundary Value Test Case		
Invalid Test Case (Min value - 1)	Valid Test Case (Min + Min, Max, -Max	Invalid Test Case (Max value - 1)
20	21,22,65,64	66

www.educba.com

Figure 1.4 shows an example of boundary value testing (educba-boundary value testing, n.d.)

1.3.5 Equivalence Partitioning Testing

Equivalence classes are created by partitioning the input space into equivalence classes according to similar characteristics or behaviors. Then, test cases that represent each equivalence class are created. By using this strategy, relevant scenarios are covered while fewer test cases are needed. It helps improve test coverage and identifies flaws relating to various input classes.

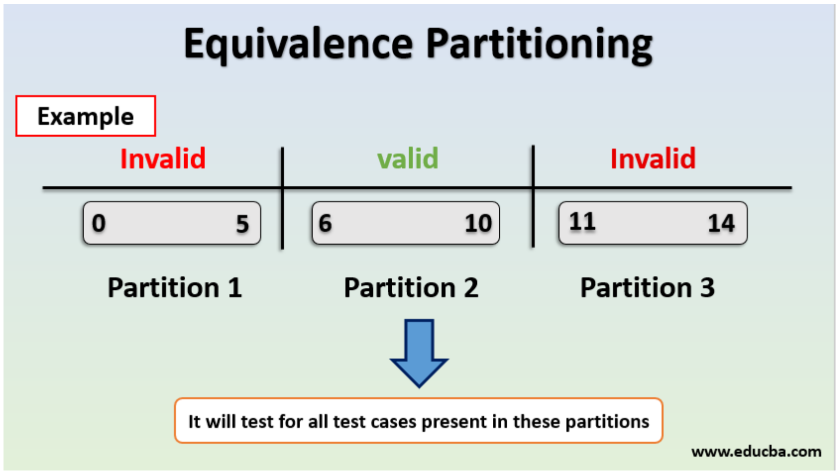


Figure 1.5 shows an illustration of equivalence partitioning (educba-equivalance partitioning, n.d.)

1.3.6 State Transition Testing

State transition testing focuses on testing the behavior of the software system as it transitions between different states or modes. It confirms that the system responds to state changes correctly and upholds data integrity across the transitions. This method is particularly helpful for systems that have well defined states and display various behaviors depending on those states.

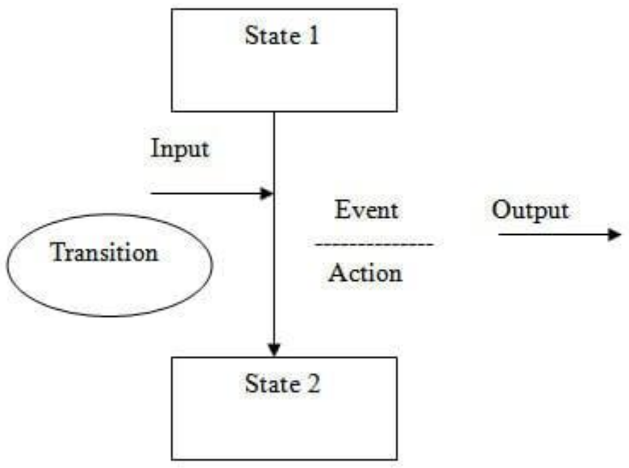


Figure 1.6 illustrates the state transition testing process (softwaretestinghelp-state transition technique, n.d.)

1.3.7 GUI Testing

In order to make sure that the system performs properly and satisfies the required specifications, GUI testing concentrates on validating the operation and behavior of the user interface elements, such as buttons, menus, forms, and windows. When performing GUI testing, testers often check that the system works as expected when various functions are carried out using the graphical interface, that input validation is carried out correctly, and that user interface buttons and components respond appropriately to user activities.

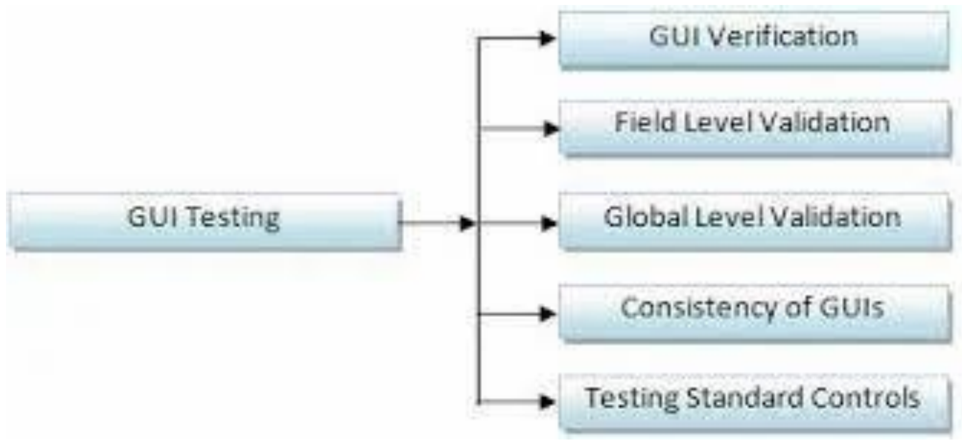


Figure 1.7 illustrates the GUI testing process (indiamart-gui testing, n.d.)

1.4 Existing Non-Functional Testing methods

1.4.1 Performance Testing

Performance testing assesses the software system's responsiveness, scalability, and stability under various workloads and stress situations. It measures things such as concurrency, resource usage, throughput, and response time. Performance bottlenecks, scalability problems, and possible areas for optimization can all be found with the aid of performance testing.

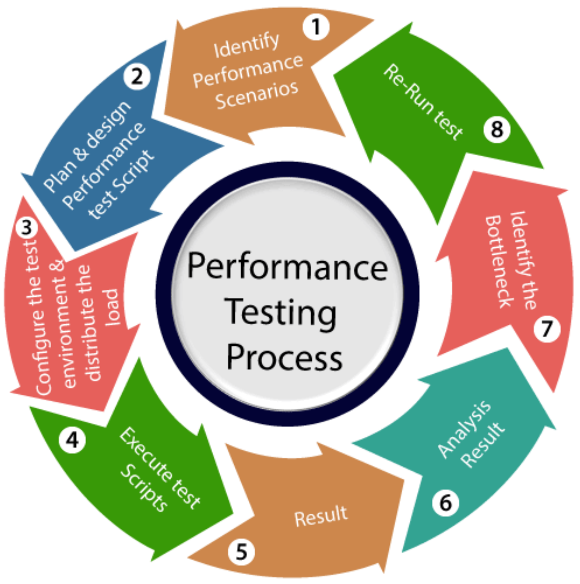

Performance Testing Process
1 Identify Performance Scenarios
2 Plan & design Performance test Script
3 Configure the test environment & distribute the load
4 Execute test Scripts
5 Result
6 Analysis Result
7 Identify the Bottleneck
8 Re-Run test

Figure 1.8 shows the performance testing process (javatpoint-performance testing, n.d.)

1.4.2 Load Testing

Load testing simulates real-world usage scenarios by subjecting the software system to anticipated loads and workloads. It checks the system's functionality both during periods of average and high load. The system's stability, resource utilization, and reaction time are evaluated through load testing to make sure it can withstand the anticipated user loads without suffering performance degradation.

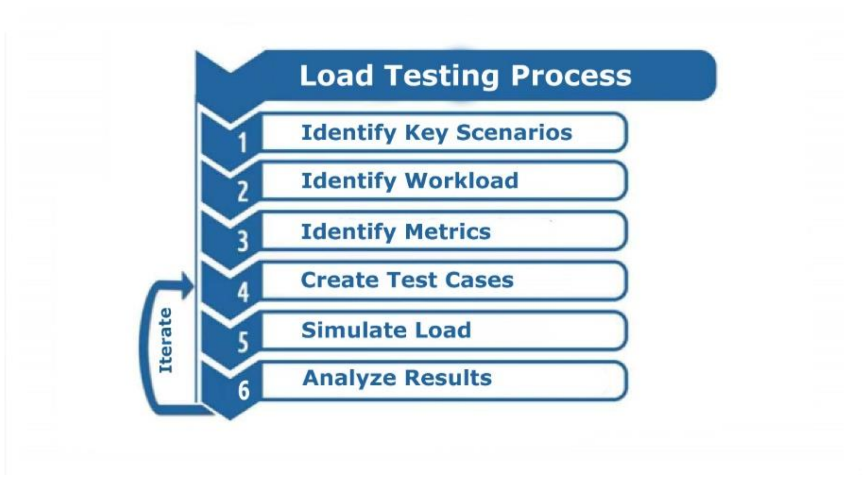


Figure 1.9 shows the Load Testing Process (techarcis-load testing, n.d.)

1.4.3 Security Testing

Security testing focuses on identifying vulnerabilities, weaknesses, and potential risks in the software system's security mechanisms. Its goal is to guarantee that the system is secure from intruders, data breaches, and other security dangers. Techniques like penetration testing, vulnerability scanning, authentication testing, and encryption testing are all part of security testing.

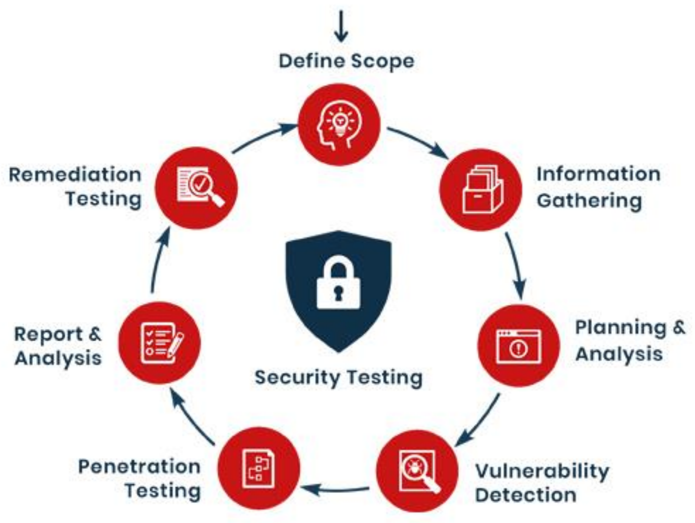


Figure 1.10 shows the Security Testing Process (testrigtechnologies-security testing, n.d.)

1.4.4 Usability Testing

Through usability testing, the software system's usability, learnability, and overall user experience are evaluated. It involves collecting feedback from sample users in order to assess the system's usability, navigation, and efficacy in accomplishing user objectives. To improve user satisfaction, usability testing identifies usability problems, design defects, and areas for improvement.

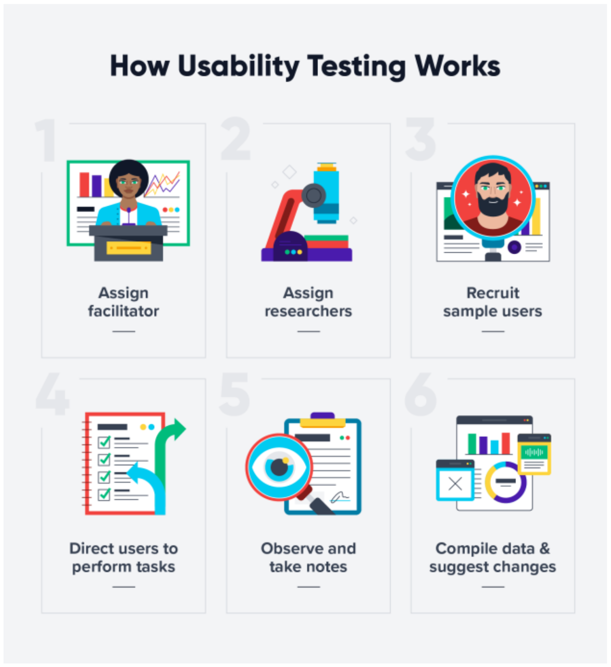

How Usability Testing Works
1
Assign facilitator
2
Assign researchers
3
Recruit sample users
4
Direct users to perform tasks
5
Observe and take notes
6
Compile data & suggest changes

Figure 1.11 shows the Usability Testing process (clevertap-usability testing, n.d.)

1.4.5 Compatibility Testing

Verifying the software system's compatibility with other platforms, operating systems, browsers, devices, or software versions is known as compatibility testing. It guarantees that the system performs accurately and consistently across a range of settings. Testing for compatibility can assist in revealing compatibility difficulties, layout or rendering issues, and functional inconsistencies between various configurations.

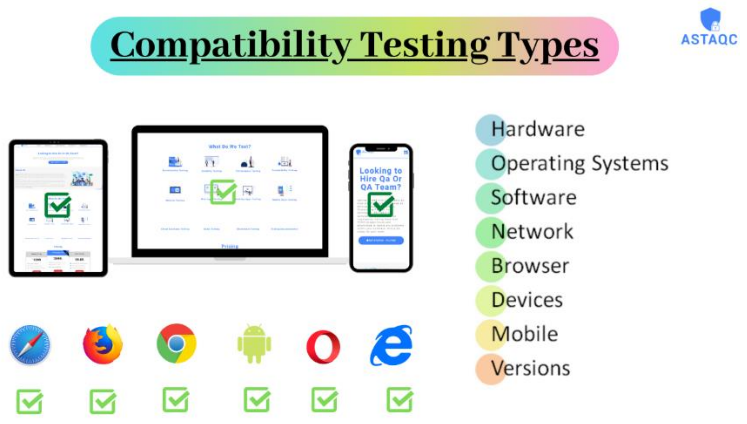


Figure 1.12 shows the different types of Compatibility testing (Linkedin-compatability testing, n.d.)

1.4.6 Stress Testing

Stress testing examines how the software system behaves and performs under conditions that are beyond its typical operational parameters. It evaluates the system's resilience, stability, and error-handling abilities when faced with heavy workloads, scarce resources, or challenging circumstances. Stress testing reveals how a system responds under pressure and whether it can make a smooth recovery.

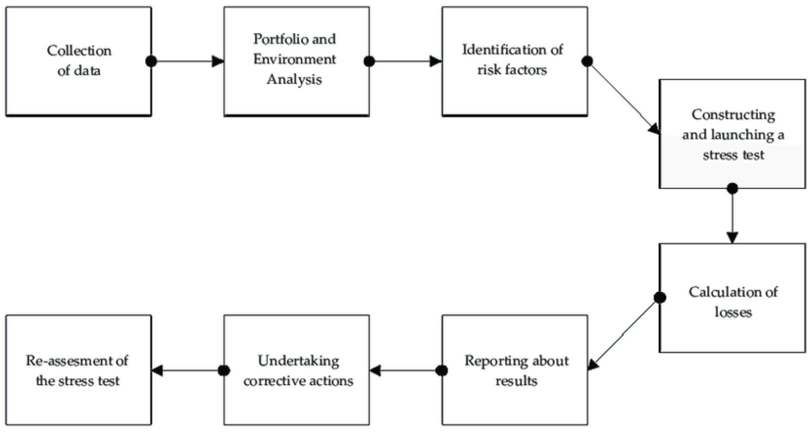

Collection of data
Portfolio and Environment Analysis
Identification of risk factors
Constructing and launching a stress test
Calculation of losses
Reporting about results
Undertaking corrective actions
Re-assesment of the stress test

Figure 1.13 shows the stages in the Stress Testing process (Mirković, 2014)

1.4.7 Reliability testing

Testing for reliability determines whether a software system can carry out its functions reliably and consistently over an extended period of time. To assess the system's resilience and ability to recover, different situations, extensive usage, and probable failures must all be applied. In order to increase system stability, reliability testing identifies potential flaws, weak spots, and places that need improvement.

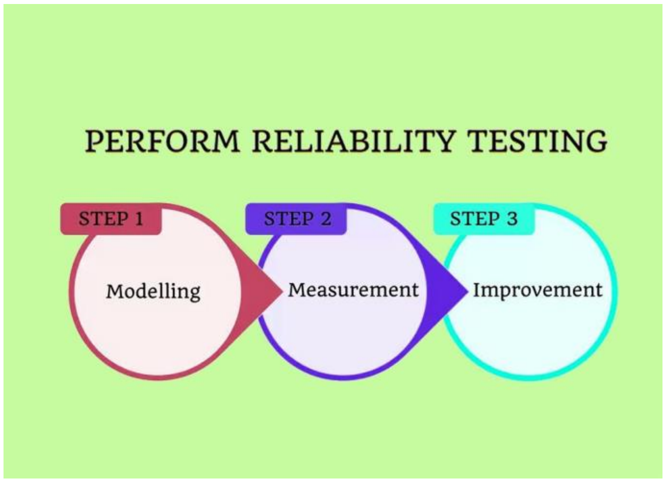


Figure 1.14 shows the process of performing Reliability Testing (appsiera-reliability testing, n.d.)

1.5 Challenges with Existing Testing Methods

1.5.1 Challenges in Existing Functional Testing methods

Functional testing methods come with their share of challenges. Unit testing has scope restrictions that frequently leave out interactions between components and real-world events.

Secondly, Integration testing faces complexities when dealing with numerous components and simulating external systems. Moreover, System testing encounters difficulties in achieving comprehensive test coverage and reproducing real-world conditions.

Moving on, Boundary value testing and equivalence partitioning testing require careful identification of boundaries and equivalence classes, respectively.

Finally, GUI testing needs significant maintenance effort due to the dynamic nature of graphical interfaces. Overcoming these challenges is crucial to ensure the effectiveness of functional testing.

In Table 1.1 we will summarize the different challenges in the existing Functional testing methods. This will help us to get a better understanding of the challenges faced when applying those methods.

Table 1.1 identifies the challenges for the existing functional testing methods.

Testing Methods	**Challenges**
Unit Testing	**Limited scope: U**nit testing focuses on individual components and may not adequately cover the interaction between components.
Integration Testing	**Complexity of integration:** Integration testing becomes challenging when there are numerous components with intricate dependencies
System Testing	**Incomplete test coverage:** It can be challenging to cover all possible scenarios and edge cases, leading to potential gaps in test coverage.
Boundary Value Testing	**Determining appropriate boundaries:** Identifying the relevant boundaries for input values can be subjective and require in-depth domain knowledge.

Equivalence Partitioning Testing	**Identifying relevant equivalence classes:** Determining appropriate equivalence classes requires a thorough understanding of the system and its input space.
State Transition Testing	**Complex state transition diagrams:** Constructing accurate state transition diagrams for complex systems can be time-consuming and error prone.
GUI Testing	**High maintenance effort:** GUI testing can be sensitive to changes in the user interface, leading to increased maintenance effort.

1.5.2 Challenges in Existing Non-Functional Testing methods

Non-functional testing methods pose their own set of challenges. Performance testing requires defining accurate performance metrics and creating realistic test scenarios.

Secondly, Load testing faces challenges in generating scalable and representative load while provisioning adequate resources. Moreover, Security testing must keep up with evolving threats and strike a balance between comprehensive testing and time constraints.

Moving on, Usability testing encounters difficulties in identifying appropriate user profiles and constructing realistic usability scenarios. Additionally, Compatibility testing involves managing diverse platforms and prioritizing compatibility issues.

Furthermore, Stress testing requires defining realistic stress levels and monitoring resource usage. Finally, Reliability testing grapples with reproducing real-world failure scenarios and accurately analyzing failure data.

Table 1.2 identifies the challenges for the existing non functional testing methods.

Testing Methods	**Challenges/Problems**
Performance Testing	**Defining performance metrics:** Determining the appropriate performance metrics and benchmarks can be challenging, as they vary based on the application type and user expectations.
Load Testing	**Generating realistic and scalable load:** Simulating a realistic load that accurately represents the expected user traffic and behavior can be challenging.

Security Testing	**Keeping up with evolving security threats and vulnerabilities**: in our modern days of advanced technology new threats are produced each day.
Usability Testing	**Identifying and recruiting appropriate user profiles:** Finding suitable user profiles for testing, ensuring they represent the target user base, can be a challenge.
Compatibility Testing	**Managing diverse platforms and configurations:** Testing compatibility across various platforms, devices, and configurations can be complex and time-consuming.
Stress Testing	**Defining realistic stress levels and thresholds:** Determining the appropriate stress levels that accurately reflect high-stress conditions for the system is essential.
Reliability Testing	**Reproducing real-world failure scenarios:** Simulating real-world usage scenarios that include system failures can be challenging during reliability testing.

1.6 Enhancement over the years

1.6.1 Functional Testing processes

Over time, improvements in technology and industry best practices have significantly improved functional testing methods. Shift-Left Testing, Agile and DevOps Integration, and automation of testing procedures are a few major improvements.

“Shift-left testing refers to the practice of testing software earlier in the development cycle than is customary, or to the left in the delivery pipeline, as opposed to the traditional practice of testing software later in the development cycle.” (Vaddadi , Thatikonda, Padthe, & Arnepalli , 2023). Early testing allows developers to find and address problems faster, resulting in more effective and efficient software development. By encouraging developers to create test cases before the code is even available for testing, shift-left testing also fosters a proactive and preventive approach to quality control.

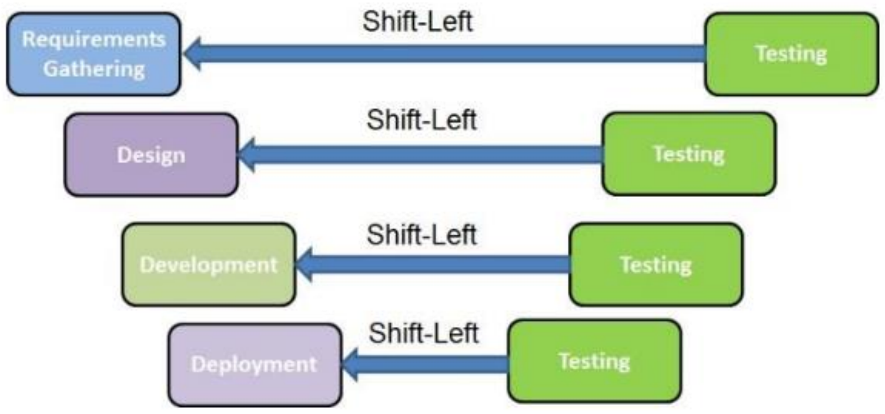


Figure 1.15 shows the Shift-Left Testing stages. (Vaddadi , Thatikonda, Padthe, & Arnepalli , 2023)

Moreover, the adoption of agile methodologies and DevOps practices has led to the integration of functional testing throughout the software development lifecycle. As a result, testing is now performed concurrently with development, enabling quicker feedback loops and iterative improvements. Additionally, the integration of Agile and DevOps methodologies offers organizations a powerful approach to handling the growing complexity associated with managing customer requirements and requests. “The combined adoption of Agile and DevOps enables organizations to cope with the increasing complexity of managing customer requirements and requests.” (Almeida, Simões, & Lopes, 2022)

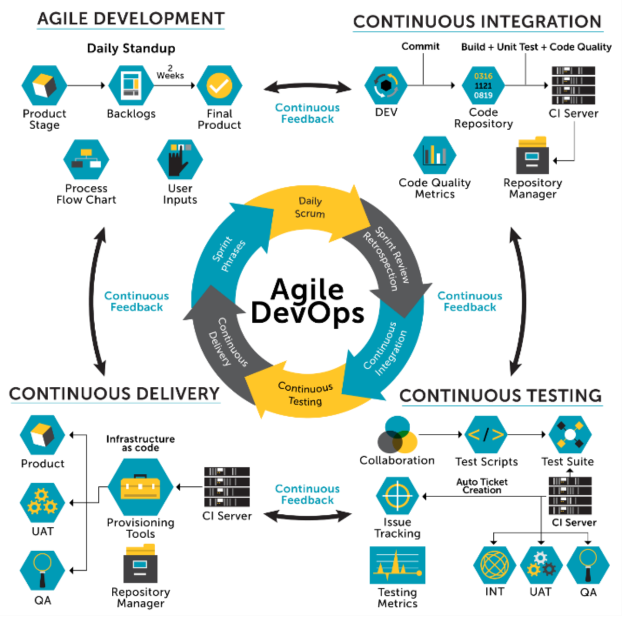


Figure 1.16 illustrates the AgileDevOps Integration (agilefirst-agile-deveops, n.d.)

Furthermore, automation of testing procedures has revolutionized functional testing. Automation of the testing procedures enables faster test execution, increased test coverage, and improved efficiency. Automated Software testing helps to decrease the workload by giving some testing tasks to the computers. “Computer systems are cheap; they are faster and don’t get bored and can work. continuously on the weekends.” (Rafi & Moses, 2011) . Also, automation enables data-driven testing, where test cases can be executed with different data inputs and configurations. This enables comprehensive testing with a wide range of data combinations, ensuring that the application handles different scenarios correctly.

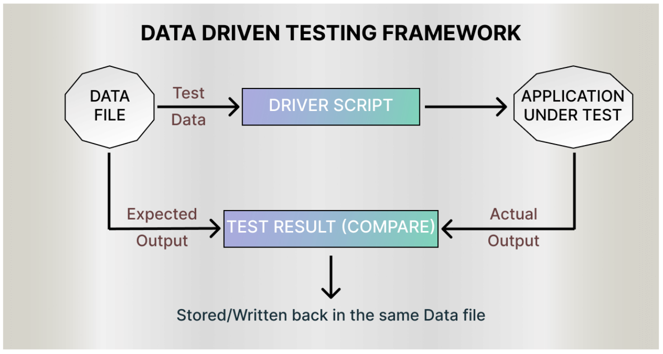


Figure 1.17 shows the Data Driven Testing process (lambdatest-data driven testing, n.d.)

1.6.2 Non-Functional Testing processes

Non-functional testing processes have also experienced significant enhancements over time. These enhancements address the evolving needs of complex software systems. Some notable enhancements include performance testing tools, cloud-based testing, and security testing techniques.

Over the years, performance testing has undergone significant advancements, and new tools have emerged to enhance the practice. With the vast range of features offered by these sophisticated performance testing tools, it is possible to do more precise measurements, thorough load generation, complex performance monitoring, and in-depth analyses of the software system. An example of the tools that emerged throughout the years is Apache JMeter. “Apache-Jmeter is a web application-based performance testing tool which is useful to examine the server performance under the heavy load (Lenka, Dey, Bhanse, & Barik, 2018). The tool mimics user loads on web applications, APIs, and databases, allowing testers to measure response times and throughput as well as identify performance bottlenecks for performance enhancement.

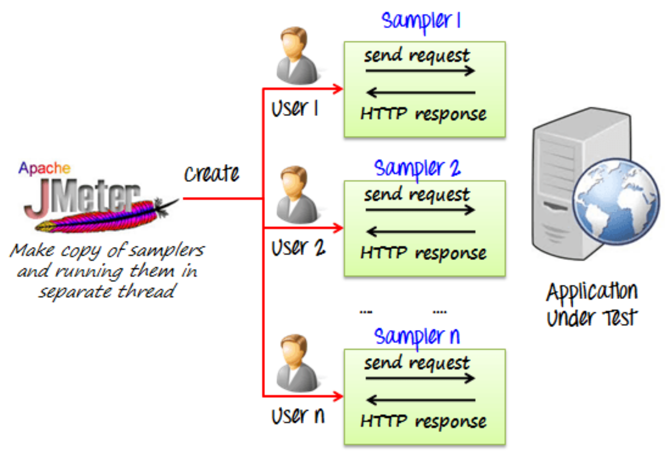


Figure 1.18 shows the process of Apache JMeter performance testing. (guru99-performance testing, n.d.)

Additionally, cloud-based testing has transformed non-functional testing practices by leveraging the benefits of cloud computing. “Cloud testing is a rapidly developing area of research in software engineering. First research works on cloud testing appeared two to three years ago.” (VILKOMIR, 2012). By utilizing cloud resources, it enables businesses to undertake performance, load, and stress testing that is scalable, flexible, and cost-effective. Testers may simply scale up or down their testing infrastructure using cloud-based testing, access a variety of virtual environments, and reduce costs by only paying for the resources they really use during testing.

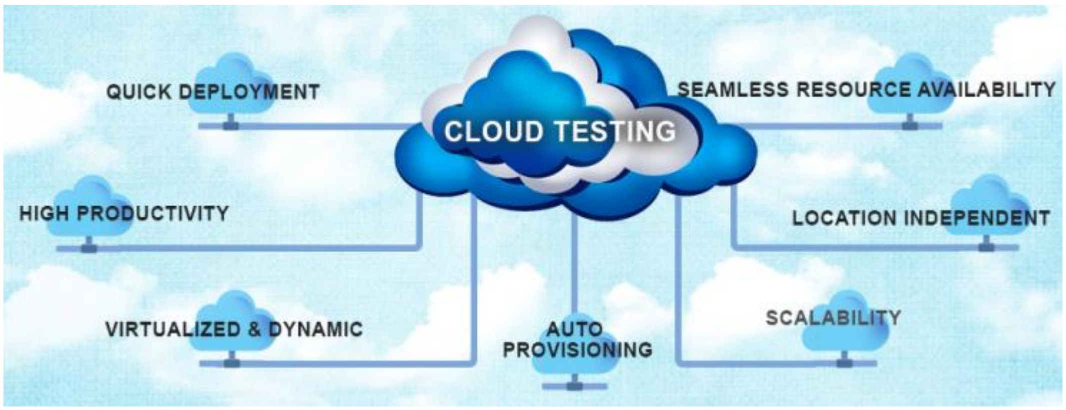


Figure 1.19 shows the benefits of cloud-based testing (v2soft-cloud testing future software testing, n.d.)

Moreover, in response to the increased importance of security in software development, security testing methodologies have developed. “Due to the openness of modern software-based systems, applying appropriate security testing techniques is of growing importance and essential to perform effective and efficient security testing.” (Felderer, et al., 2016). In order to successfully identify and address security vulnerabilities, several new techniques and technologies have surfaced. These methods include dynamic security testing and penetration testing.

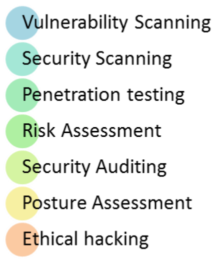


Figure 20 shows the various Security testing techniques (guru99-security testing, n.d.)

1.7 Functional Testing Vs Non-Functional Testing

First, Functional testing and Non-Functional testing are both crucial in ensuring the overall quality and reliability of software systems. Functional testing validates the system's behavior and functional requirements to make sure it carries out its intended functions correctly. Non-functional testing, on the other hand, addresses important factors of the system including performance, security, usability, and compatibility of the software system.

Moreover, Functional testing confirms the system's fundamental functionality, whereas non-functional testing assesses the system's usability, dependability, and overall performance from the user's point of view.

Hence, by doing both types of testing, organizations can make sure that their software systems not only meet functional requirements but also provide a high-quality user experience, operate well under a variety of conditions, and are safe and compatible. This comprehensive approach to testing helps meet user expectations and enhance user satisfaction.

Table 1.3 showcases the differences between functional testing and non-functional testing.

Parameters	**Functional Testing**	**Non-Functional Testing**
Test Focus	Verifying functional requirements and behavior	Assessing performance, security, usability, etc.
Test Objective	Ensure correct functioning of the system	Evaluate system attributes beyond functionality
Test Execution	Carried out before non-functional testing	Carried out after functional testing
Test Requirements	Usually, it’s easy to define requirements functional requirements	Usually, it is hard to define non-functional requirements
Example of a test case	Check the functionality of generating a receipt	The receipt page should load in maximum duration of 5 seconds.

2. Literature Review

In this section, we delve into a thorough examination of the existing research and scholarly work on functional and non-functional testing techniques. We can learn important things about how testing procedures have changed and advanced through time. Additionally, we can have a better understanding of the developments made, the difficulties encountered, and the best practices used in the field of functional and non-functional. This literature review is an essential step in our understanding of the core ideas, approaches, and modern developments in functional and non-functional testing techniques.

2.1 Functional testing background study

First, (Bühler & Wegener, 2008) emphasizes the significance of evolutionary testing in automating functional testing. The use of evolutionary testing methodologies improves the testing process overall by enabling more rapid and accurate defect discovery. It transforms testing objectives into search problems and utilizes evolutionary computation techniques to solve them. The paper focuses on applying evolutionary testing to automate functional testing, particularly in the context of complex automotive systems. The results of the paper indicate that evolutionary testing can be successfully applied to automate functional testing in complex systems, leading to improved fault detection and enhancing the testing process.

Next, (Shi , 2010) addressed the challenges faced in software migration and implementation projects within industrial firms, emphasizing the significance of functional testing. The study emphasizes how black-box approaches, such as Equivalence-Classes Partitioning, Boundary-Value Analysis, and Error Guessing, may be used to create efficient and cost-effective test cases for real-world business scenarios. According to the paper, using test tools can help software functional testing be even more effective. Also, industrial companies can get more out of their functional testing efforts by using strategies like functional clustering and interrelated test case mapping.

Moving further, (Ferrer, Kruse, Chicano, & Alba, 2015) addresses the problem of dynamic test sequences from a formal specification as a complementary approach to traditional functional testing methods. The study proposed to use search-based approaches to generate minimal test sequences that meet the required coverage criteria. The study shows the superiority of the search-based approaches in generating test sequences with optimal coverage. The research results expand testing methodologies and offer suggestions for enhancing the effectiveness and precision of functional testing techniques using formal specifications.

Additionally, (Singh & Sharma, 2015) addressed the need for efficient software testing to identify errors and defects. The study concentrated on specification-based testing, which creates test cases based on the functional specifications listed in the Software Requirement Specification. The objective of the study was to automate the process of generating functional test cases. The solution proposed is a framework for automated generation of use case diagrams. Activity diagrams are then used to generate functional test cases using the depth-first search technique. The study's findings suggest that the work and time needed for software testing can be drastically decreased by automatically creating functional test cases and use case diagrams. Software testers can concentrate on other crucial parts of testing by automating these operations, which improves productivity and the overall quality of the software system.

2.2 Non-Functional Testing background study

(Arteaga, Contreras, & Barrientos, 2019) presents a methodology primarily for non-functional testing while also acknowledging the significance of the various testing phases for mobile applications. The paper proposes a framework specifically for non-functional testing. The framework encompasses the identification of relevant non-functional requirements, the definition of test case scenarios (TCS), and the selection of appropriate tools for executing these TCS. Expert surveys and deployment of the framework to an actual mobile application were both done to validate the suggested framework. The outcomes of the validation process show that the framework is successful in bringing the non-functional testing process' complexity down.

Moreover, (Harman, Jia, & Zhang, 2015) addressed the concern of growing range of non-functional properties being addressed using SBST. However, the proportion of papers dedicated to this topic is decreasing. The paper also outlines the limited study work on Search Based Energy Testing (SBET) that has been done and investigates alternative energy measuring methods that could be used as fitness functions, along with any difficulties that might arise. The FiFiVerify tool challenge is presented at the end of the paper, which ends with a positive outlook on the future of SBST. The potential for interesting developments in the near future is indicated by the authors' belief that basic FiFiVerify tools are currently within the reach of the research community.

Furthermore, (Ribeiro & Travassos, 2016) addresses the increasing importance of Non-Functional Requirements (NFRs) in ensuring the success of software systems and the corresponding need for effective testing. Two structured literature studies were used in this study's analysis, which identified a number of NFRs that are not sufficiently addressed by current approaches to software testing. Another notable finding is that the majority of the identified software testing approaches lack empirical evaluation. Hence, absence of empirical evaluation hinders the understanding of the effectiveness and limitations of these approaches in real-world scenarios. These findings offer opportunities for further research in the field.

Finally, (Camacho, Marczak, & Cruzes, 2016) highlights several challenges faced by agile teams in testing non-functional requirements. Factors such as priority, cost, and time pressure have been identified as persistent issues since traditional waterfall development. To reduce the challenges associated with non-functional testing in agile teams, the paper suggests several possible solutions. The first approach is to include analysis of non-functional aspects during project meetings, such as project inception and sprint planning. Additionally, non-functional reflection during user story development can also help in identifying and addressing non-functional requirements.

3. Risk of not performing either one of the testing

In software development, testing plays a crucial role in ensuring the quality, reliability, and performance of a software system. To reduce risks and deliver a reliable software product, functional and non-functional testing are both necessary. This section will discuss the risks associated with not performing either one of these testing types and highlight the potential consequences.

3.1 Risks of Not Performing Functional Testing

Functional testing makes sure that the software complies with the specified functional specifications and carries out the anticipated actions. Failure to perform adequate functional testing can lead to several risks such as defects and malfunctioning of software system, incomplete implementation, and poor user experience.

First, there could be a risk of defect causing malfunction in the system. Without adequate functional testing, the software may contain undetected flaws that cause malfunctions, errors, or unexpected behavior while the system is in use. Customers may become dissatisfied, their trust may be lost, and there may be financial losses as a result.

Additionally, there is a risk of incorrect or implementation of the software system. Functional testing helps validate the implementation of desired functionalities. Without enough testing, the chance of an inadequate or inaccurate implementation increases, endangering the system's intended functionality and behavior. This could lead to improper data processing, wrong outcomes, or unstable systems.

Finally, there is a risk of non-compliance with requirements. Functional testing ensures that the software complies with the expected functionality described in the project scope and satisfies the specified requirements. The risk of not meeting these standards rises when functional testing is neglected. Without enough testing, it's possible to forget to implement important features, functionalities, or business rules.

3.2 Risks of Not Performing Non-Functional Testing

Non-functional testing is crucial for evaluating the system's performance, security, and other non-functional attributes. Failure to conduct comprehensive non-functional testing poses several risks such as performance issues, security vulnerabilities, and compatibility and interoperability problems.

Firstly, the risk of system performance issues rises. Non-functional testing techniques like stress testing and load testing aid in locating performance bottlenecks, scalability constraints, and response time problems. Without sufficient performance testing, the system could experience performance issues, such as slow response times, high resource consumption, or system crashes, especially when it is being used heavily.

Secondly, security vulnerabilities risk is introduced. Non-functional testing, including security testing and penetration testing, aims to identify vulnerabilities and weaknesses in the system's security measures. Hence, neglecting security testing can expose the software to potential security breaches, data leaks, unauthorized access, and damage to the system's integrity and reputation.

Finally, there could be compatibility and interoperability issues in the system. Non-functional testing examines how well the system works with various hardware, software, and operating system configurations. Compatibility problems can arise from a failure to conduct compatibility testing, making the software incompatible with environments or creating interoperability challenges when connecting with other systems or software components.

3.3 Summary of identified risks of not performing either testing.

Risk	Consequences
Defects and Malfunctioning	- Customer dissatisfaction - Potential financial losses
Incomplete implementation	- Incorrect data processing - Inaccurate results
Non-compliance with Requirements	- Legal consequences, compliance issues, reputational damage for the organization

Performance Issues	- Slow response times - High resource consumption
Security Vulnerabilities	- Data leaks - Unauthorized access
Compatibility and Interoperability issues	- System crashes when exchanging data or communicating with other environments

4. Conclusion and Future Work

In this study, we have explored various aspects of software testing, with a particular focus on functional and non-functional testing. We have highlighted the significance of both types of testing in ensuring the quality and success of software systems.

Functional testing plays a critical role in verifying the system's adherence to specified requirements, identifying defects, and ensuring the correct implementation of desired functionalities.

On the other hand, non-functional testing addresses the performance, security, usability, and other critical aspects of the software system. It aims to assess the system's behavior under various conditions and its ability to meet non-functional requirements.

Moving on, we have identified the risks associated of not performing either testing technique. Hence, Organizations must implement a thorough testing approach that includes both functional and non-functional testing to reduce these risks. To achieve this, it is necessary to carefully plan, design, implement, execute, and analyze test cases. It also calls for the proper application of the necessary testing techniques, tools, and methodologies.

In addition to the key findings presented above, this study also identifies potential avenues for future work and research in the field of software testing. As software systems continue to evolve with new technologies and paradigms such as cloud computing, Internet of Things (IoT), and blockchain, there is a need to adapt testing practices accordingly. Future study could focus on examining the requirements and obstacles related to testing these new technologies and creating specialized testing methods to ensure their performance, security, and reliability.

By addressing these future research directions, we can advance the field of software testing and contribute to the ongoing improvement of software quality assurance practices.

5. References

Vaddadi , S. A., Thatikonda, R., Padthe, A., & Arnepalli , P. R. (2023). Shift-Left Testing Paradigm Process Implementation for Quality of Software Based on Fuzzy. *Research Square*, 1-19.

agilefirst-agile-deveops. (n.d.). Retrieved from agilefirst.com: https://agilefirst.io/agile-devops/

Almeida, F., Simões, J., & Lopes, S. (2022). Exploring the Benefits of Combining DevOps and Agile. *Future Internet*, 14-63.

appsiera-reliability testing. (n.d.). Retrieved from appsierra.com: https://www.appsierra.com/blog/reliability-testing-in-software-testing

Arteaga, F., Contreras, F., & Barrientos, A. (2019). A Framework for Non-Functional Testing Process of Mobile Applications. *IEEE XXVI International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 1-4.

Bühler, O., & Wegener, J. (2008). Evolutionary functional testing. *Computers & Operations Research*, 3144 – 3160.

Camacho, C. R., Marczak, S., & Cruzes, D. (2016). Agile Team Members Perceptions on Non-Functional Testing. *11th International Conference on Availability, Reliability and Security*, 582-589.

clevertap-usability testing. (n.d.). Retrieved from clevertap.com: https://clevertap.com/blog/usability-testing/

educba-boundary value testing. (n.d.). Retrieved from educba.com: https://www.educba.com/boundary-value-testing/

educba-equivalance partitioning. (n.d.). Retrieved from educba.com: https://www.educba.com/equivalence-partitioning/

Felderer, M., Buchler, M., Johns, M., Brucker, A., Breu, R., & Pretschner, A. (2016). Security Testing: A Survey. *Advances in Computers*, 1-44.

Ferrer, J., Kruse, P., Chicano, F., & Alba, E. (2015). Search based algorithms for test sequence generation in functional testing. *Information and Software Technology*, 419-432.

guru99-performance testing. (n.d.). Retrieved from guru99.com: https://www.guru99.com/jmeter-performance-testing.html

guru99-security testing. (n.d.). Retrieved from guru99.com: https://www.guru99.com/what-is-security-testing.html

Harman, M., Jia, Y., & Zhang, Y. (2015). Achievements, open problems and challenges for search based software testing. *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 1-12.

indiamart-gui testing. (n.d.). Retrieved from indiamart.com: https://www.indiamart.com/proddetail/gui-testing-7370046891.html

Javatpoint-Integration Testing. (n.d.). Retrieved from Javatpoint.com: https://www.javatpoint.com/integration-testing

javatpoint-performance testing. (n.d.). Retrieved from javatpoint.com: https://www.javatpoint.com/performance-testing

Katalon-Unit testing. (n.d.). Retrieved from katalon.com: https://katalon.com/resources-center/blog/unit-testing-integration-testing *lambdatest-data driven testing*. (n.d.). Retrieved from lambdatest.com: https://www.lambdatest.com/learning-hub/data-driven-testing

Lenka, R. K., Dey, M. R., Bhanse, P., & Barik, R. K. (2018). Performance and Load Testing: Tools and Challenges. *International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering - (ICRIEECE)*, 2257-2262.

Linkedin-compatability testing. (n.d.). Retrieved from Linkedin.com: https://www.linkedin.com/pulse/compatibility-testing-types-process-tools-benefits-avanish-pandey/

Mirković, V. (2014). STRESS TESTING IN FINANCIAL INSTITUTIONS. *Creative Commons Attribution-ShareAlike 4.0 International*, 88-15.

Rafi, D. M., & Moses, K. R. (2011). Automated Software Testing A Study of State Practice. *Blekinge Institute of Technology*, 1-124.

Ribeiro, V., & Travassos, G. (2016). Testing Non-Functional Requirements: Lacking of Technologies or Researching Opportunities? *Brazilian Symposium on Software*, 226-240.

Shi , M. (2010). Software Functional Testing from the Perspective of Business Practice. *Computer and Information Science*, 49-52.

Singh, A., & Sharma, S. (2015). Functional Test Cases Generation Based on Automated Generated Use Case Diagram. *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, 105-110.

softwaretestinghelp-state transition technique. (n.d.). Retrieved from softwaretestinghelp.com: https://www.softwaretestinghelp.com/state-transition-testing-technique-for-testing-complex-applications/

softwaretestinghelp-system-testing. (n.d.). Retrieved from softwatestinghelp.com: https://www.softwaretestinghelp.com/system-testing/

techarcis-load testing. (n.d.). Retrieved from techarcis.com: https://www.techarcis.com/load-testing-in-agile-environment-do-you-know-it-all-2/

testrigtechnologies-security testing. (n.d.). Retrieved from testrigtechnologies.com: https://www.testrigtechnologies.com/service/security-testing/

v2soft-cloud testing future software testing. (n.d.). Retrieved from v2soft.com: https://www.v2soft.com/blogs/cloud-testing-future-software-testing

VILKOMIR, S. (2012). CLOUD TESTING: A STATE-OF-THE-ART REVIEW. *Information & Security An International Journal*, 213-222.

6. Appendix

6.1 Performance testing objectives

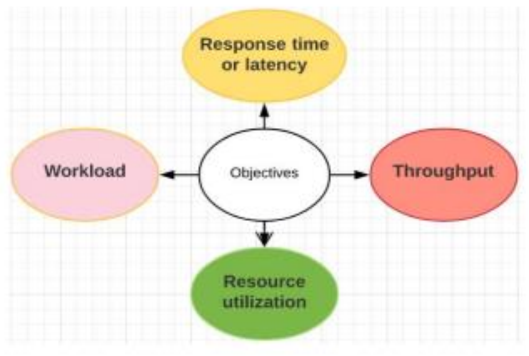


Fig 6.1 shows the performance testing objectives (Lenka, Dey, Bhanse, & Barik, 2018)

6.2 Evolutionary testing process

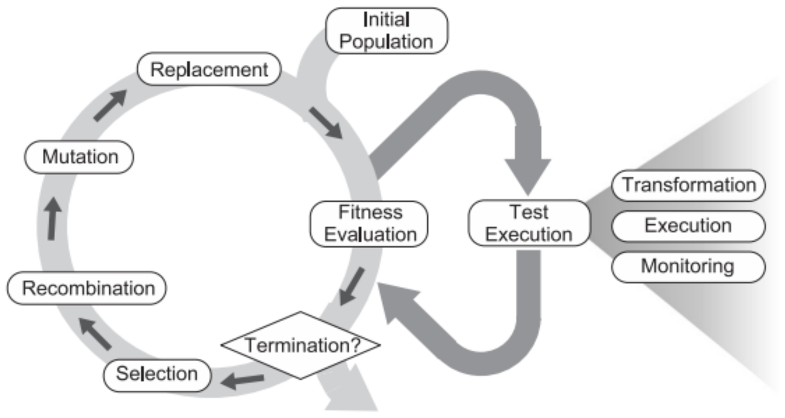


Fig 6.2 shows the evolutionary testing process (Bühler & Wegener, 2008)

6.3 Testing dimensions

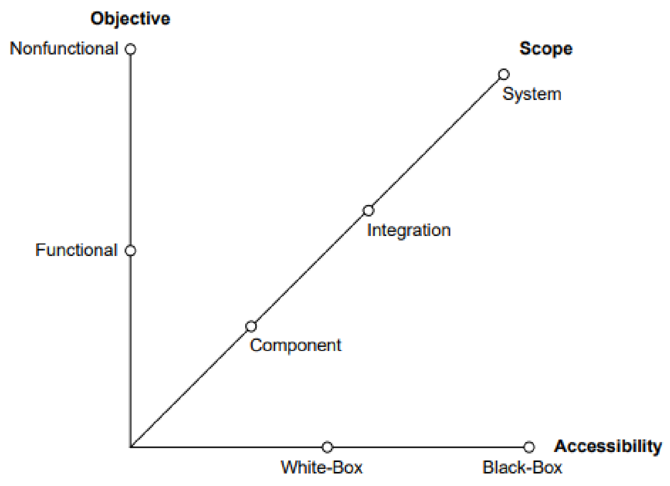
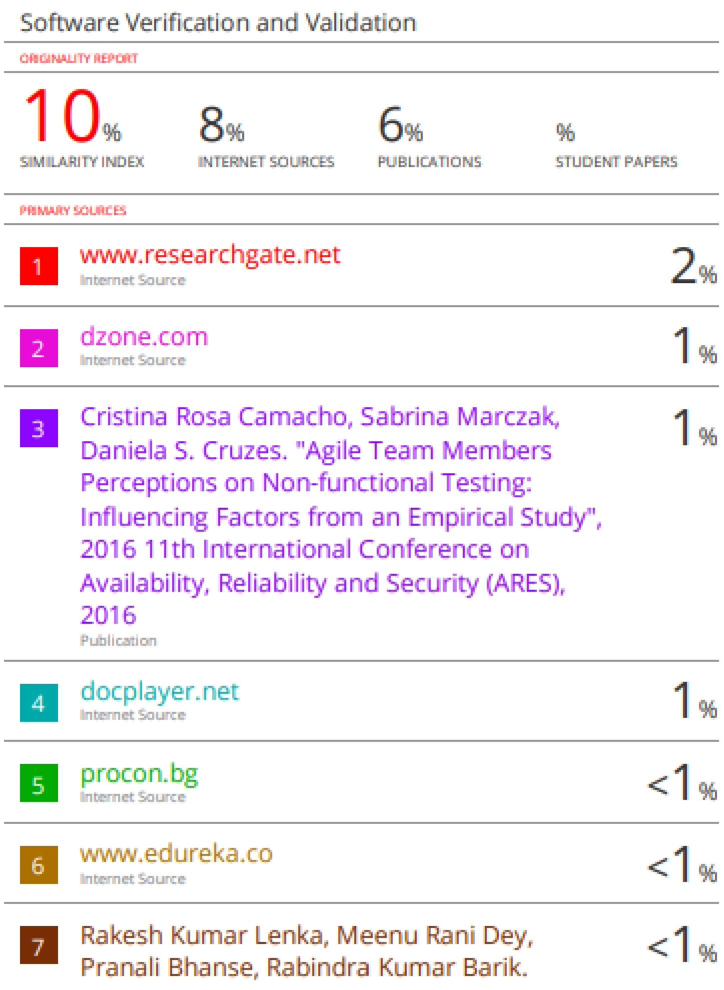


Figure 6.3 shows the testing Dimensions Objective, Scope, and Accessibility

(Felderer, et al., 2016)

6.4 Turnitin report similarity index



Software Verification and Validation

ORIGINALITY REPORT

10%	8%	6%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	2%
2	dzone.com Internet Source	1%
3	Cristina Rosa Camacho, Sabrina Marczak, Daniela S. Cruzes. "Agile Team Members Perceptions on Non-functional Testing: Influencing Factors from an Empirical Study", 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016 Publication	1%
4	docplayer.net Internet Source	1%
5	procon.bg Internet Source	<1%
6	www.edureka.co Internet Source	<1%
7	Rakesh Kumar Lenka, Meenu Rani Dey, Pranali Bhanse, Rabindra Kumar Barik.	<1%

	"Performance and Load Testing: Tools and Challenges", 2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE), 2018 Publication	
8	www.mdpi.com Internet Source	<1 %
9	Yudong Tao, Tianyi Wang, Anchen Sun, Shahid S. Hamid, Shu‐Ching Chen, Mei‐Ling Shyu. "Florida public hurricane loss model: Software system for insurance loss projection", Software: Practice and Experience, 2022 Publication	<1 %
10	eprints.whiterose.ac.uk Internet Source	<1 %
11	www.newestech.com Internet Source	<1 %
12	Rosmiati Rosmiati. "Analisis Dan Pengujian Sistem Menggunakan Black Box Testing Equivalence Partitioning", Jurnal Sains Komputer dan Teknologi Informasi, 2021 Publication	<1 %
13	aisel.aisnet.org Internet Source	<1 %
14	annamalaiuniversity.ac.in Internet Source	<1 %

15	academicworks.cuny.edu Internet Source	<1%
16	w3softech.com Internet Source	<1%
17	www.citrenz.ac.nz Internet Source	<1%
18	www.iaeng.org Internet Source	<1%
19	Rijwan Khan. "Deep Learning System and It's Automatic Testing: An Approach", Annals of Data Science, 2021 Publication	<1%
20	docshare.tips Internet Source	<1%
21	idr.mnit.ac.in Internet Source	<1%
22	riuma.uma.es Internet Source	<1%
23	testingwebperformance.blogspot.com Internet Source	<1%
24	vdocuments.mx Internet Source	<1%
25	Harsh Bhasin, Neha Singla, Shruti Sharma. "Cellular automata based test data	<1%

Marking Rubric: ASSESSMENT FORMS

Assignment Assessment Form

Place it on the last page of your submission

Name: Ahmed Hassan Mohammed Salem

Student ID: 1191102340

Subject: Software Verification and Validation (TSE3251)

Date: 27th June 2023

	Excellent	**Good**	**Average**	**Poor**	**Not acceptable**	**Total**
1. Introduction to your assignment: 1.1. Introduction (What are Functional and Non-Functional Testing Techniques)	The introduction concisely outlines ideas that will be discussed in this assignment. The assignment statement encompasses the core of the assignment.	The introduction outlines most of the ideas that will be discussed in this assignment. The assignment statement covers most of the core of the assignment.	The introduction outlines some of the ideas that will be discussed in this assignment. The assignment statement covers some of the core of the assignment.	The introduction outlines only the general idea of what will be discussed in this assignment. The assignment statement covers a little of the core of the assignment.	The introduction does not outline the assignment or is not found at all. The assignment statement does not cover the core of the assignment.	**/10**
1.2 Objective of Doing Functional and Non-Functional Testing 1.3 Existing testing method for both (List down and explain each in detail)	The objective and method demonstrate mastery of the subject and are explicitly and strongly connected to both the points covered in the subject.	The objective and method demonstrate above average understanding of both the points covered in the subject.	The objective and method demonstrate an average understanding of both the points covered in the subject.	The objective and method demonstrate a passing understanding of both the points covered in the subject.	The objective and method demonstrate a poor understanding of both the points covered in the subject.	**/20**

1.4 Challenges/Problems with existing testings methods (you can use tabular format to differentiate them) 1.5 Enhancements in each Testing Processes over the Years 1.6 Functional Testing Vs Non-Functional Testing (explain why both are necessary?)	The Three (3) points are well-researched with substantial literature, with clear and detailed discussion based on references of more than one perspective of the issue.	The Three (3) points are sufficiently researched, with clear discussion based on references of more than one perspective of the issue.	The Three (3) points are researched to a limited extent, with some discussion based on references of more than one perspective of the issue.	The Three (3) points are researched to a limited extent, with little to some discussion based on references of only one perspective of the issue.	The Three (3) points are researched to a limited extent, with limited discussion of references with no focus on the issues.	**/30**
2. Literature Review of previous work 3. Risk of not performing either one of the testing	The Lit Review and Risk are well-researched with substantial literature, with clear and detailed discussion.	The Lit Review and Risk are sufficiently researched, with clear discussion.	The Lit Review and Risk are researched to a limited extent, with some discussion.	The Lit Review and Risk are researched to a limited extent, with little to some discussion.	The Lit Review and Risk are researched to a limited extent, with limited discussion.	**/20**
Conclusion and Future Work A conclusion and future work are provided, summarizing the assignment's salient points.	The conclusion clearly and sufficiently summarises the main points of the assignment.	The conclusion clearly but only adequately summarises the main points of the assignment	The conclusion adequately summarises only some of the points of the assignment.	The conclusion poorly summarises some of the main points of the assignment.	The conclusion lacks any focus to summarise the main points of the assignment. May contain new points that were not mentioned previously in the assignment.	**/10**
The **referencing** system is applied correctly and consistently.	The assignment has no citation errors and no errors on the works cited page.	The assignment has 1-2 citation errors and no errors on the works cited page.	The assignment has 1-2 citation errors and 1-2 errors on the works cited page.	The assignment has more than 2-4 citation errors and more than 2-4 errors on the works cited page.	The assignment has more than 4 citation errors and more than 4 errors on the works cited page.	**/5**
Quality and Clarity The assignment is written in correct,	The assignment demonstrates logic and clarity in writing with	The assignment demonstrates logic and clarity in writing with	The assignment demonstrates logic and clarity in writing with	The assignment demonstrates some logic and clarity in	The assignment demonstrates a lack of logic and clarity in	**/5**

clear and concise **English** and is correctly formatted.	clear and concise English and is correctly formatted all the time. Contains no more than 2 minor errors (typos, spelling, subject-verb agreement, punctuation, etc.).	clear and concise English and is correctly formatted most of the time. Contains no more than 2 major errors; no more than 4 minor errors.	clear and concise English and is correctly formatted some of the time. Contains no more than 4 major errors; no more than 6 minor errors.	writing with clear and concise English and is not correctly formatted most of the time. Contains no more than 6 major errors; no more than 8 minor errors.	writing with clear and concise English and is not correctly formatted most of the time. Contains no more than 6 major errors; more than 8 minor errors.	

COMMENTS:

OVERALL GRADE: -------/100 * 0.3 = (------------)