International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
***Article Received:** 26 August 2023 **Revised:** 09 October 2023 **Accepted:** 27 October 2023*

A Structured Cloud-Based Software Testing Model with a Case Study Implementation

Santhosh S[1*], Narayanaswamy Ramaiah[2]
[1]Computer Science and Engineering
Faculty of Engineering and Technology, Jain (Deemed-to-be University)
Bangalore, India
Santhu87@ymail.com
[2]Computer Science and Engineering
Faculty of Engineering and Technology, Jain (Deemed-to-be University)
Bangalore, India
Dr.narayanaswamyramaiah@gmail.com

Abstract— Cloud-based testing methodologies were gaining significant popularity and adoption in the software testing industry. Cloud-based testing offers several advantages, such as scalability, flexibility, cost-effectiveness, and the ability to access a wide range of testing tools and environments without the need for extensive infrastructure setup. Cloud testing methods are having challenges with respect to testing priority, practical use cases, performance, lengthy test time, integrating and streamlining, data security, etc. since they are addressing specific purposes. To address these challenges, there is a need for a structured testing model with respect to the cloud environment. This article proposes a new structured cloud-based testing model for enhancing the testing service in the cloud environment. The proposed model addresses the order of testing and the priority, data security, and performance by using Smoke and Sanity testing methods.

Keywords- Cloud-Based Testing, Cloud-based SDLC, Structured Testing, Smoke Testing, Sanity Testing, Software Engineering**.**

I. INTRODUCTION

Many businesses spend around 40% of their resources on software testing, which results in higher costs. Software testing is an important component of the Software Development Lifecycle. Because of the resource availability and the varieties of cloud services, testing software that runs in the cloud is more simple as well as more challenging. Performance, security, workload, and adaptability tests are all part of cloud-based testing, and several tools and techniques have been created to test cloud systems. Test cases and test scripts are used for testing to validate the requirements. The various cloud-based testing methods available in the current industry are as follows:

- **Cloud-Based Test Execution**: In this approach, testing is performed on virtual machines or containers hosted in the cloud. Testers can execute test scripts and test scenarios on various configurations, operating systems, and browsers without setting up physical hardware. Cloud-based test execution provides rapid scalability, allowing teams to run tests concurrently on multiple virtual machines, thus reducing testing time.
- **Cloud-Based Test Environments**: Instead of maintaining on-premises testing environments, teams leverage cloud infrastructure to set up and manage test environments. These environments can be quickly provisioned and de-provisioned, which is especially useful for parallel testing or simulating real-world scenarios with varying system configurations.
- **Device Cloud Testing**: For mobile and cross-browser testing, device cloud testing services offer a wide range of real devices hosted in the cloud. Testers can perform compatibility testing across different devices and operating system versions without having physical access to each device.
- **Cloud-Based Test Data Management**: Cloud-based test data management solutions provide secure and scalable storage for test data. This ensures that test data is easily accessible to the testing team while maintaining data privacy and compliance.
- **Cloud-Based Performance Testing**: Cloud platforms offer the ability to conduct load and performance testing by simulating a large number of virtual users from various geographic locations. Cloud-based performance testing allows for stress testing of applications without putting a strain on internal infrastructure.
- **Cloud-Based Security Testing**: Security testing in the cloud can involve using cloud-based security scanning tools and services to perform vulnerability assessments, penetration testing, and security code reviews on the application under test.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

- **Continuous Integration and Continuous Testing (CI/CT) with Cloud**: Teams leverage cloud-based Continuous Integration (CI) and Continuous Testing (CT) pipelines to automate the build, deployment, and testing processes. Cloud-based CI/CT enables faster feedback loops and supports Agile and DevOps practices.
- **Smoke Testing**: Performing smoke testing on every build is essential to identify defects in the early stages. It serves as the final check before the software build is deployed to the system stage. Smoke tests are a mandatory step for each build that undergoes testing, including new developments and major or minor system releases. To initiate smoke testing, the QA team needs to verify the correct build version of the application being tested. This straightforward process efficiently assesses the application's stability in a short amount of time. By incorporating smoke tests, testing efforts can be minimized, ultimately leading to improved application quality. [13].

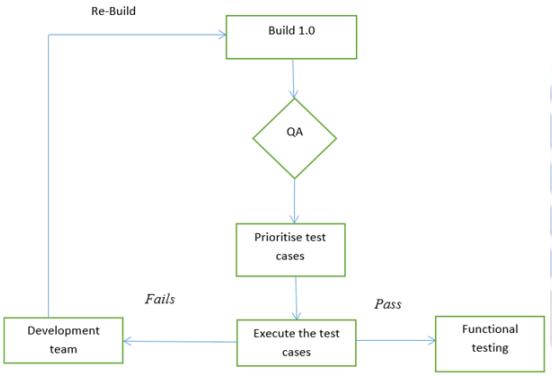

Figure. 1: Smoke Testing Lifecycle [13]

- **Sanity Testing**: Sanity testing, also referred to as surface-level testing, is a technique used to evaluate whether newly added features or functionality in software applications have successfully addressed any related bugs [14]. This approach serves as a fundamental and preliminary test to determine the correct functioning of a software application or its specific components. It is essential to perform sanity testing in various scenarios, including:
- When minor changes are made to the application's code.
- After adding new features that are ready to be integrated into the software application.
- Following a series of regression tests and the generation of a new build.
- After applying bug fixes.
- Prior to the production deployment of the application.

Before integrating features into the software application.

II. LITERATURE REVIEW

According to authors [1], Cloud testing presents unique challenges that need to be acknowledged and addressed to fully realize the benefits it can offer to the testing process. The complexity and various dimensions involved require a deep understanding of testing priorities, enabling clarity and confidence when operating in cloud environments. While on-premise testing practices still have their relevance, they must be reassessed and aligned with the cloud testing strategy since legacy approaches may not be suitable in the cloud context. Testing within Continuous Integration/Continuous Deployment (CI/CD) pipelines can significantly enhance efficiency, but certain testing aspects like Security, Accessibility, and Exploratory testing may still require manual efforts. Quality is a collective responsibility, and cloud environments offer opportunities for increased collaboration and instilling quality as a core behavior. To maintain credibility and value in the cloud landscape, quality engineering skills must expand to encompass proficiency in multiple coding languages, architectural awareness, and a comprehensive understanding of cloud principles. Understanding and safeguarding critical business processes and underlying technologies are crucial for ensuring business continuity in case of failures. Although cloud platforms promise many advantages over traditional systems, realizing those benefits requires careful consideration of numerous factors, with quality assurance playing a vital role in the process.

[2] A novel cloud-based development and testing framework is being introduced, leveraging the advantages offered by various cloud computing models and technologies. The main goal of this framework is to seamlessly integrate and offer relevant tools that streamline software development and testing activities. Moving forward, the framework prototype will be meticulously designed and developed, with a focus on implementation. Additional scenarios will be explored to thoroughly assess the feasibility and advantages of the cloud-based development and testing framework. These scenarios will serve as practical use cases to gauge the framework's performance and capabilities.

[3] The article explores the domain of automated testing in cloud environments. After analyzing existing automated cloud test technologies, the authors propose a novel automation software test technology that utilizes a virtualization test environment. They emphasize that automated function testing and performance testing are particularly well-suited for cloud environments. The flexible and virtualized nature of the environment allows for concurrent test execution, efficient cross-platform testing, and seamless integration with automated testing tools. The paper outlines the entire test process and provides guidance on developing and executing test scripts. As

cloud computing technology continues to evolve, the demand for cloud testing providers is increasing. However, the paper also points out certain challenges. These challenges include the need to address concerns related to mechanics, data security, and other security issues that could impact both the actual environment and the virtual environment used for testing.

[4] The primary objective of the proposed study is to develop a technology acceptance model that applies an expert system based on the codified knowledge of experts using artificial intelligence techniques. This model aims to facilitate decision-making processes related to cloud adoption specifically for software testing purposes. By leveraging artificial intelligence techniques, the expert system will be able to offer decision support to software development organizations (SDOs). The support will come in the form of various predictors and determinants that will guide SDOs toward making informed decisions about whether to adopt cloud-based solutions for their software testing needs. Ultimately, the proposed model seeks to provide valuable insights and guidance to SDOs, helping them navigate the complexities of cloud adoption and optimize their software testing processes.

[5] In this paper, the authors present a multi-layer model for a cloud-based online software testing platform to address the challenges of high costs, lengthy test times, and complex test implementations in traditional software testing. The proposed model leverages cloud computing to provide a more efficient and user-friendly testing solution. The multi-layer model encompasses various components, including an Infrastructure as a Service (IaaS) platform, a Software as a Service (SaaS) platform, a self-help service portal for users, and an operation maintenance portal for administrators. Through these integrated components, the platform offers the automatic creation of virtual machines (VMs), remote access to test tools, and online quality assurance services, catering to the fundamental testing needs of users. They implemented a real software online testing platform based on this model. The platform has been tested to assess the scale of VMs it can support. The results indicate that the average provisioning time of a virtual machine generally increases as the number of VM requests grows. While the concurrent startup of VMs affects compute and controller nodes, the network node remains relatively unaffected. They have also successfully applied this model to several actual projects for customers, validating its applicability and stability. By adopting this cloud-based testing platform or its specific techniques, they have effectively addressed the challenges of traditional software testing and achieved more efficient and reliable testing processes.

[6] Cloud testing is a rapidly growing field within cloud computing. This article presents a comprehensive review of different testing methods that are applicable and beneficial in a cloud environment. Additionally, the paper provides a detailed description of the steps to be followed in the testing process, offering insights into various testing scenarios. Furthermore, the paper offers an overview of various testing tools available in the market. Depending on the specific requirements of end-users, they can choose from a range of tools suited for testing in a cloud computing environment.

[7] Author discusses various cloud testing techniques and examines commercial testing tools available in the market. Despite being in the early stages of cloud testing, the paper identifies some of the challenges associated with this domain through an analysis of research papers. Building on the insights gained from the challenges, the authors plan to develop a new testing framework. This framework aims to address the unique requirements and complexities of cloud testing, thereby enhancing the quality and reliability of cloud-based applications. By continuously improving testing methodologies and tools for cloud environments, organizations can maximize the benefits of cloud computing while ensuring robust and thoroughly tested applications.

[8] The article introduces a novel approach for test case selection and prioritization in a distributed cloud environment, based on multi-objective optimization. The proposed Resemblance-Based Cluster Head (RBCH) algorithm is utilized to select the Cluster Head (CH) by considering the overall similarity between test cases. Furthermore, the Distance-Based Transposition (DBT) technique is proposed to prioritize the optimal test case clusters in the distributed environment. This approach efficiently reduces time consumption during the testing process by minimizing the number of iterations in the test case search. Also demonstrates the effectiveness of the proposed approach, showcases higher fault detection rates, prioritization accuracy, and lower execution times compared to existing prioritization techniques.

[9] The study examines existing research on cloud-based performance testing and highlights the strengths and weaknesses of current approaches. The paper also compares P-TaaS with traditional performance testing methodologies. Overall, the paper sheds light on the importance and potential of P-TaaS in enhancing the efficiency and effectiveness of performance testing in cloud computing environments.

[10] In this article, the authors explored the various strategies for software testing and the diverse approaches to artificial intelligence in the testing domain. Highlighted the primary benefits that arise from incorporating artificial intelligence into the software testing process. It also showcased a few examples of artificial intelligence-driven tools that have been purposefully designed for software testing. In essence, this article aims to provide insights into the significance of software testing in the digital era and how the integration of artificial intelligence can enhance testing efficiency and effectiveness, thereby ensuring the robustness and reliability of digital products and applications.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

A structured process model, such as the CSLCP model, discussed in the article [11] may assist SMEs in producing affordable cloud-based software of a high caliber. A case study was used as evidence to fully describe and show the actions involved in the software life cycle. Training various team members on the CSLCP model may enhance software dependability, save costs, and shorten the time needed to deploy software in SMEs when team members work on several projects and perform diverse roles. Moreover, since the CSLCP model is compatible with CMMI levels two and three, it improves the quality, cost, and schedule of development processes, thereby raising the maturity level of SMEs.

For developing and building cloud computing systems from the viewpoints of both vendors and customers, a model known as the Cloud Computing Development Life Cycle (CCDLC) has been introduced. The conventional software engineering process models' limitations and restrictions are addressed by the CCDLC paradigm [12]. The conventional SDLC process as well as other significant processes are modified.

Challenges:

The existing solutions study provides insight that the Cloud testing methods are having challenges with respect to testing priority, practical use cases, performance, lengthy test time, integrating and streamlining, data security, etc. since they are addressing specific purposes. So there is a need for a structured software testing model for the cloud environment to perform the testing activities simply and smoothly.

III. PROPOSED METHOD

The proposed Structured Cloud-Based testing model is shown in Figure. 2. It contains nine phases of testing with respect to infrastructure, continuous integration, repository, security, other vulnerability assessment, and performance. After all the testing, the reporting will be done with the remediation plan for the failed test cases. Then the remediation process takes place by the developers in consultation with the testing team. Finally, the deployment testing will be conducted to confirm the completion of testing and approval.

Phase 1 (Cloud-Based Infrastructure): The security testing process begins within the cloud-based infrastructure, where various testing components are provisioned and configured. This infrastructure could include virtual machines, containers, and cloud services required for security testing activities.

Phase 2 (CI/CT Environment): Within the cloud environment, Continuous Integration and Continuous Testing (CI/CT) pipelines are set up. These pipelines are responsible for automating security testing processes, including running security tests at various stages of the software development lifecycle.

Phase 3 (Source Code Repository): The application's source code is stored in a cloud-based repository, such as GitHub or GitLab. The code repository serves as the central location for the application's codebase and allows seamless integration with the CI/CT pipeline.

Phase 4 (Cloud-Based Security Scans): As part of the CI/CT process, the application's source code and artifacts undergo security scanning using cloud-based security testing tools. These tools perform various security assessments, such as Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Software Composition Analysis (SCA).

Phase 5 (Vulnerability Assessment): The security scanning tools identify potential vulnerabilities, weaknesses, and security issues within the application's code and dependencies. These vulnerabilities are categorized based on severity and potential impact.

Phase 6 (Performance Testing): A testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload.

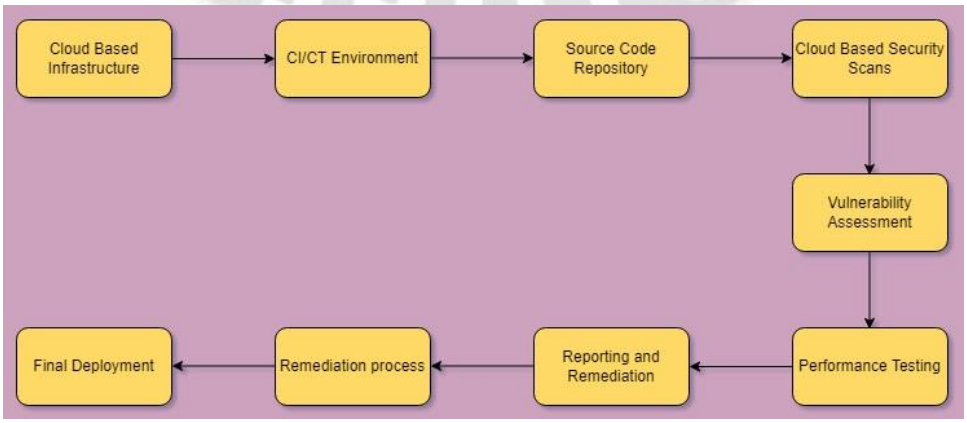

Figure. 2: The Structured Cloud-Based Testing Model.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

Phase 7 (Reporting & Remediation): The CI/CT pipeline generates comprehensive security reports that outline the identified vulnerabilities and their associated risks. The development and security teams use these reports to prioritize and address security issues.

Phase 8 (Remediation Process): The development team addresses the security vulnerabilities by implementing necessary fixes and improvements. The fixes may involve code changes, updates to dependencies, or configuration changes.

Phase 9 (Final Deployment): Once the security issues are addressed, the application is retested to ensure the effectiveness of the remediation efforts. The application, now with improved security, can be deployed to the target environment(s) with confidence.

IV. CASE STUDY

The case study "Howsort-Algorithm Visualizer Application" has its main objective to create a tool to facilitate the flow of learning much easier and reduce the time spent on understanding through smaller logical chunks that amount to the complete complex algorithmic logic. This application aims its users to be ones that getting started with learning the popular sorting algorithms out there for upgrading themselves.

A. Cloud Requirement Analysis and SLA

As per the Software requirements specification, the services required to develop the application are as follows:

1. Infrastructure:
    a. EC2 instance m4.xlarge (16GiB RAM, 4 vCPUs, 750MBPS bandwidth, 0.394USD/Hour)
    b. S3 storage (10GB)
    c. Elastic Beanstalk service
2. Monitoring and Management service
    a. AWS Cloud watch service
    b. AWS Cloud Trail service
3. Development tools and application services
    a. AWS command line
    b. AWS API gateway
    c. AWS CodeBuild
    d. AWS CodeDeploy
    e. AWS CodeCommit
4. Other requirements
    a. Android studio
    b. Draw.io web tool
    c. Figma and Balsamiq tools

B. Design

Figure. 3 represents the architecture and the flow diagram of the application. It describes how the user navigates through the app and it describes each functionality like where the user has to give the inputs and on the execution of those inputs where the outputs are to be displayed. The architecture also describes the screens, animation section, and control that are designed in the app and shows all the features and functionality that are there in the application.

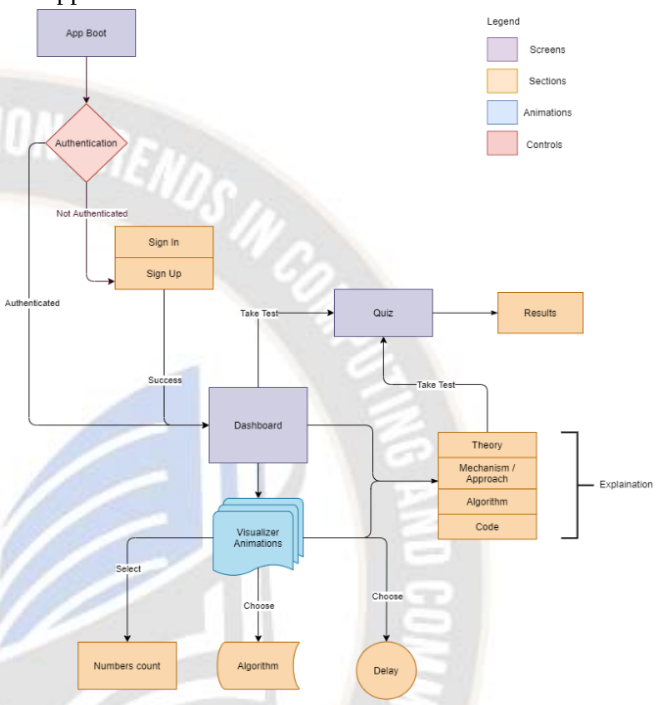

Figure. 3: Architecture and flow diagram

C. Implementation

Figure 4 is the sign-up or the registration screen where the user creates an account for the first time and these details are stored in the authentication and real-time database of Firebase.

Figure 5 screenshot of the application has theory screen showing one of the sorting algorithms – bubble sort. The contents of the theory screen have been gathered and optimized to make users easily understand what an algorithm is all about. It consists of definition section where, a brief explanation about the algorithm such as what does it do, what approach does it follow, where it is used, what mechanism and technique that it uses, and more.

Finally, the code section displays and helps in seeing all the conditional statements being displayed on the visualizer screen (Figure 6). Code is shown in three different languages, C for more traditional way of code, Java for better performant code, and python for most easier and less code.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

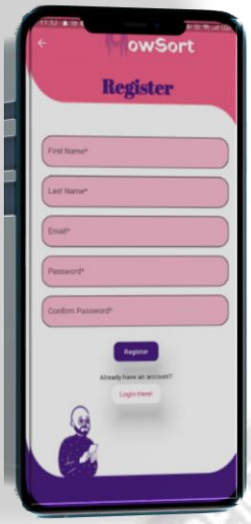

Figure 4: Application register (sign-up) screen

Figure 5: Theoretical explanation of an Algorithm

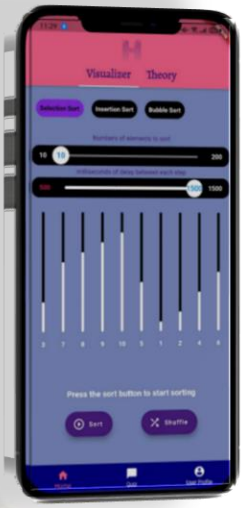

Figure 6: Application Visualizer

D. Testing

The case study application was tested using the structured Cloud-Based Software Testing model to provide a simple and smooth testing activity. The smoke testing and Sanity Testing methods were adopted for deriving the test cases for the application testing in the cloud environment.

i. Continuous Integration (CI) and Continuous Testing (CT) with Cloud involve automating the build, testing, and deployment processes using cloud-based resources. The below table [Table 1] provides sample test cases for Continuous Integration and Continuous Testing with Cloud:

Table 1: Test cases derived for CI/CT testing.

Test Case #1: CI/CT Environment Setup				
Objective: To ensure the CI/CT environment is properly configured and ready for automated testing.				
Test ID	**Description**	**Test Steps**	**Expected Result**	**Pass/Fail**
TC-CI/CT-001	CI/CT Environment Setup	1. Verify that the cloud-based CI/CT infrastructure is provisioned with the necessary resources and tools.	The CI/CT environment should be ready for test execution with appropriate build agents, testing tools, and testing frameworks installed.	Pass
		2. Validate that the source code repository is correctly integrated with the CI/CT pipeline.	The CI/CT pipeline should be triggered automatically whenever code changes are pushed to the repository.	Pass
		3. Check if the CI/CT environment is capable of building the application from the source code.	The CI/CT environment should be able to build the application successfully without any build errors.	Pass
		4. Verify that the environment allows parallel execution of test cases across multiple configurations or devices (if applicable).	The CI/CT environment should support concurrent test execution, allowing for faster feedback on test results.	Fail
		5. Ensure that proper access controls are implemented to protect sensitive data and configurations in the cloud-based CI/CT environment.	Access to the CI/CT environment should be restricted to authorized personnel only.	Pass
Test Case #2: Automated Test Execution				
Objective: To verify that automated tests are executed successfully in the CI/CT environment.				

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
***Article Received:** 26 August 2023 **Revised:** 09 October 2023 **Accepted:** 27 October 2023*

Test ID	Description	Test Steps	Expected Result	Pass/Fail
TC-CI/CT-002	Automated Test Execution	1. Trigger the CI/CT pipeline by committing code changes to the source code repository.	The CI/CT pipeline should automatically start the build and testing process upon detecting code changes.	Pass
		2. Verify that the build process is successful without any errors or warnings.	The build should complete successfully, generating a deployable artifact.	Pass
		3. Confirm that the automated tests, including unit tests, integration tests, and acceptance tests, are executed as part of the CI/CT process.	All automated tests should be executed without any failures, providing comprehensive test coverage.	Pass
		4. Check that test execution results are recorded and stored for future reference.	Test results, including pass/fail status and detailed logs, should be stored for analysis and review.	Pass
		5. Validate that the CI/CT pipeline can handle concurrent test execution and manage test environments effectively (if applicable).	The CI/CT environment should efficiently handle multiple test executions and manage test environments efficiently to avoid conflicts.	Pass
		6. Ensure that any notifications or alerts are generated and sent in case of test failures or issues.	The CI/CT pipeline should send notifications to relevant stakeholders when tests fail or when issues occur during the CI/CT process.	Pass
Test Case #3: Continuous Deployment				
Objective: To ensure the successful deployment of the application using the CI/CT pipeline.				
Test ID	**Description**	**Test Steps**	**Expected Result**	**Pass/Fail**
TC-CI/CT-003	Continuous Deployment	1. Verify that the CI/CT pipeline is configured to automatically deploy the application to the target environment after successful testing.	The application should be deployed to the target environment automatically without manual intervention once all tests pass.	Pass
		2. Check that the deployment process is repeatable, consistent, and can be rolled back if needed.	The CI/CT pipeline should ensure consistent and reliable deployments, allowing rollbacks in case of any issues with the new version.	Pass
		3. Validate that proper versioning and tagging mechanisms are in place for the deployed artifacts.	Deployed artifacts should be accurately versioned and tagged to track changes and facilitate traceability.	Pass
		4. Ensure that the application is successfully deployed in the target environment(s) specified in the CI/CT pipeline configuration.	The application should be accessible and functional in the specified target environment(s) after the deployment is completed.	Pass
		5. Verify that any necessary database migrations or data changes are applied during the deployment process.	Database schema changes and data migrations should be executed seamlessly during the deployment process, ensuring data consistency.	Pass
		6. Check that the CI/CT pipeline reports the status of the deployment process, including success or failure.	The CI/CT pipeline should provide clear and accurate information about the status of the deployment process, helping identify any deployment issues.	Pass

ii. Security testing is a critical aspect of software testing, and it involves evaluating a system to identify potential vulnerabilities and weaknesses. The following are some of the examples:

- SQL Injection: Test to check if the application is vulnerable to SQL injection attacks by attempting to input malicious SQL statements in input fields. For

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
***Article Received:** 26 August 2023 **Revised:** 09 October 2023 **Accepted:** 27 October 2023*

example, entering ' OR 1=1; -- in a login form to bypass authentication.

- Cross-Site Scripting (XSS): Test to verify if the application can protect against XSS attacks. Input various scripts, such as <script>alert('XSS');</script>, in different input fields to see if the application properly sanitizes and escapes user inputs.
- Password Strength: Test to ensure that the system enforces strong password policies, such as minimum length, complexity, and expiration, to prevent easy password guessing or cracking.
- Session Management: Test to check the security of session management, ensuring that session tokens are random, not exposed in URLs, and invalidated after logout or inactivity.
- Authentication Bypass: Attempt to bypass the authentication mechanism by using common default usernames/passwords, blank passwords, or by manipulating cookies or hidden fields.
- File Uploads: Check if the application restricts the types of files that can be uploaded and verifies file contents to prevent malicious files (e.g., malware, scripts) from being uploaded.
- Error Handling and Information Leakage: Test to ensure that the system does not reveal sensitive information in error messages or stack traces that could be exploited by attackers.
- Access Control Testing: Test to verify if users have appropriate permissions and access levels and ensure they cannot access unauthorized areas or perform actions outside their privilege scope.
- Encryption and Data Protection: Verify that sensitive data like passwords, credit card details, etc., are stored securely using strong encryption algorithms.
- Denial of Service (DoS) and Distributed Denial of Service (DDoS) Attacks: Test to check if the application can withstand DoS and DDoS attacks by simulating high loads and excessive requests.
- Business Logic Vulnerabilities: Analyze the application's business logic to identify any flaws that could be exploited, such as unauthorized transactions or privilege escalations.
- Third-Party Component Security: Assess the security of third-party libraries and components used in the application to ensure they are free from known vulnerabilities.
- Mobile Application-Specific Security Tests: For mobile apps, test for jailbreak/root detection, insecure data storage, and insecure communication.
- Input Validation: Verify that all user inputs are properly validated to prevent malicious data from being processed.
- API Security Testing: Test the security of APIs to ensure they are protected against unauthorized access and data manipulation.

Table 2: Test cases derived for security testing.

Test Case #1: SQL Injection				
Objective: To verify that the application is protected against SQL injection attacks.				
Test ID	**Description**	**Test Steps**	**Expected Result**	**Pass/Fail**
TC-Sec-001	SQL Injection Prevention	1. Navigate to the input field susceptible to SQL injection.	The input field is accessible and ready to receive user input.	Pass
		2. Enter the following SQL injection attempt: **' OR 1=1; --**	The application should not execute the injected SQL statement and should respond appropriately, such as denying access or providing an error message.	Pass
		3. Observe the application's response.	The application should handle the input properly, without executing the malicious SQL code. There should be no SQL-related errors or leakage of sensitive information.	Pass
		4. Repeat steps 2 and 3 for other input fields vulnerable to SQL injection.	The application should protect against SQL injection attacks consistently across all vulnerable input fields.	Pass
		5. Attempt SQL injection using other SQL injection techniques (e.g., UNION-based, time-based, etc.) if applicable.	The application should detect and prevent various SQL injection techniques, demonstrating robust security against such attacks.	Pass
		6. Verify logs and error messages (if any) to ensure no sensitive information is exposed.	Error messages or logs should not reveal any database-related information or other sensitive details that could aid attackers in their exploits.	Pass

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

Test Case #2: Cross-Site Scripting (XSS)				
Objective: To verify that the application is protected against Cross-Site Scripting (XSS) attacks.				
Test ID	**Description**	**Test Steps**	**Expected Result**	Pass
TC-Sec-002	XSS Prevention	1. Navigate to the input field or page susceptible to XSS attacks.	The input field or page is accessible, and it should be ready to receive user input without rendering any injected scripts.	Pass
		2. Enter the following XSS script: **<script>alert('XSS');</script>**	The application should handle user input properly and prevent the execution of the script. It should either display the input as plain text or sanitize the input by converting special characters to their HTML entities.	Pass
		3. Observe the application's response.	The application should not execute the injected script but should handle the input safely.	Pass
		4. Repeat step 2 and 3 for other input fields or pages vulnerable to XSS attacks.	The application should protect against XSS attacks consistently across all vulnerable input fields or pages.	Pass
		5. Attempt different types of XSS attacks, such as reflected, stored, and DOM-based XSS (if applicable).	The application should prevent various XSS techniques, ensuring users cannot execute malicious scripts on other users' browsers or steal their cookies or session information.	Pass
		6. Verify logs and error messages (if any) to ensure no sensitive information is exposed.	Error messages or logs should not reveal any security-related details or sensitive information that could aid attackers in their exploits.	Pass
Test Case #3: Password Strength				
Objective: To verify that the application enforces strong password policies.				
Test ID	**Description**	**Test Steps**	**Expected Result**	Pass
TC-Sec-003	Password Strength Policy	1. Access the user registration or password change form.	The registration or password change form is accessible.	Pass
		2. Attempt to set a new password with a weak password (e.g., "password" or "123456").	The application should validate the password against the password policy and inform the user that a stronger password is required.	Pass
		3. Attempt to set a new password that meets the minimum length requirement but lacks complexity (e.g., "abcde").	The application should validate the password against the password policy and inform the user that a stronger password with a combination of uppercase, lowercase letters, numbers, and special characters is required.	Pass
		4. Set a password that meets all the password policy requirements (e.g., "P@ssw0rd123").	The application should accept the strong password and store it securely.	Pass
		5. Verify that password complexity and length requirements are enforced during password change as well.	The application should consistently enforce password policies during both registration and password change processes.	Pass
Test Case #4: Session Management				
Objective: To verify the security of the application's session management mechanism.				
Test ID	**Description**	**Test Steps**	**Expected Result**	Pass
TC-Sec-004	Session Token Generation	1. Log in to the application with valid credentials.	The application should generate a secure session token for the user's session.	Pass
		2. Observe the session token details.	The session token should be random, long enough, and should not contain any predictable patterns.	Pass

		3. Attempt to manipulate the session token manually or via script.	The application should detect any unauthorized tampering with the session token and should invalidate the session, forcing the user to log in again.	Pass
		4. Log in with invalid credentials or without authentication.	The application should not generate a session token for unauthorized access attempts.	Pass
		5. Verify the session expiration time and activity timeout.	The application should invalidate the session and log out the user automatically after a certain period of inactivity or upon reaching the session expiration time.	Pass
		6. Check for session fixation vulnerabilities by attempting to set a user's session ID manually and then log in using that session ID.	The application should generate a new session ID upon successful authentication, preventing session fixation attacks.	Pass
		7. Logout from the application.	The application should invalidate the session and remove any associated session token, ensuring that the user is logged out securely.	Pass
		8. Verify if "Remember Me" functionality, if available, uses a secure method of storing persistent session information.	The "Remember Me" functionality should store persistent session information securely, using techniques like secure cookies or long-lived tokens.	Pass
Test Case #5: API Security Testing				
Objective: To verify the security of the application's APIs.				
Test ID	**Description**	**Test Steps**	**Expected Result**	Pass
TC-Sec-005	API Authentication	1. Identify the APIs that require authentication (e.g., OAuth, API keys, JWT, etc.).	The application should have proper authentication mechanisms in place for all sensitive APIs.	Pass
		2. Attempt to access authenticated APIs without proper authentication credentials.	The application should deny access to unauthorized users and respond with an appropriate error message, such as a 401 Unauthorized status code.	Pass
		3. Test the validity of authentication tokens or keys used in API requests.	The application should verify the authenticity of the provided authentication tokens or keys, and reject any invalid or expired ones.	Pass
		4. Test the usage of HTTPS for secure communication.	Sensitive data transmitted through APIs should be encrypted using HTTPS to prevent eavesdropping and man-in-the-middle attacks.	Pass
		5. Verify if sensitive data is being exposed in API responses or logs.	API responses and logs should not contain any sensitive information that could be exploited by attackers.	Pass
		6. Test for potential API vulnerabilities, such as SQL injection, XSS, and CSRF vulnerabilities, by passing manipulated input to the APIs.	The application's APIs should be protected against common security vulnerabilities, and input validation should be in place to prevent malicious data from being processed.	Pass
		7. Check for API rate limiting and usage quotas, if applicable.	The application should enforce rate limiting and usage quotas on APIs to prevent abuse and potential DoS attacks.	Pass
		8. Test the API for CORS (Cross-Origin Resource Sharing) misconfigurations.	The application should have proper CORS configurations to restrict cross-origin requests and prevent unauthorized access from other domains.	Pass
		9. Verify if APIs return appropriate error responses for different scenarios (e.g., 400 Bad Request, 403 Forbidden, 500 Internal Server Error).	The application should provide meaningful and consistent error responses to help users and developers identify and handle issues effectively.	Pass

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 10
Article Received: *26 August 2023* ***Revised:*** *09 October 2023* ***Accepted:*** *27 October 2023*

iii. Performance testing

Performance testing in the cloud involves evaluating the performance characteristics of an application or system under varying workloads using cloud-based resources.

Table 3: Test cases for performance testing

Test Case #1: Load Testing				
Objective: To evaluate the application's performance under different load levels.				
Test ID	**Description**	**Test Steps**	**Expected Result**	**Pass/Fail**
TC-PT-001	Load Testing	1. Identify target scenarios representing peak, normal, and low load conditions.	The scenarios should be representative of real-world usage patterns.	Pass
		2. Set up cloud-based load testing tools or services to simulate virtual users generating traffic as per the defined scenarios.	The load testing tools should be able to simulate the desired number of virtual users concurrently accessing the application.	Pass
		3. Run the load tests and monitor key performance metrics such as response time, throughput, and server resource utilization.	The application's response time should remain within acceptable thresholds under various load conditions.	Fail
		4. Gradually increase the load to reach peak levels and observe how the application behaves under stress.	The application should be able to handle peak loads without significant degradation in performance.	Pass
		5. Analyze test results to identify performance bottlenecks, if any, and capture data on resource utilization, database queries, and other resources.	Performance bottlenecks, if present, should be identified and documented with possible remediation steps.	Pass
		6. Determine if the application's performance meets the predefined performance objectives and Service Level Agreements (SLAs).	The application should meet the performance objectives, and SLAs related to response times and other critical performance metrics should be achieved.	Pass
		7. Repeat load tests with different user scenarios and workloads to validate the application's scalability and ensure that performance remains consistent under varying loads.	The application should demonstrate scalability, with consistent performance across different user scenarios and workloads.	Pass
Test Case #2: Stress Testing				
Objective: To assess the application's stability and responsiveness under extreme load conditions.				
Test ID	**Description**	**Test Steps**	**Expected Result**	**Pass/Fail**
TC-PT-002	Stress Testing	1. Identify scenarios to simulate heavy user loads, beyond the system's intended capacity.	The scenarios should stress the system to observe its behavior under extreme load conditions.	Pass
		2. Configure cloud-based stress testing tools to apply load beyond the system's capacity, gradually increasing the number of virtual users.	The stress testing tools should be able to generate heavy loads and simulate concurrent virtual users.	Pass
		3. Run the stress tests and monitor the application's behavior, including response time, errors, and resource usage during tests.	The application should remain stable, and critical errors should be captured and analyzed.	Pass
		4. Analyze the performance data to identify potential points of failure and areas of improvement.	Points of failure and areas for improvement should be identified, and appropriate actions should be recommended to enhance stability.	Pass
		5. Validate that the application's response time remains acceptable even	The application's response time should not exceed defined thresholds, and essential functionality should remain operational.	Pass

		under extreme stress, and critical operations remain functional.		
		6. Conduct post-stress testing analysis to identify potential performance degradation, memory leaks, or system instability issues.	The post-stress analysis should highlight any performance degradation, memory leaks, or system instability that occurred during the stress tests.	Pass
Test Case #3: Endurance Testing				
Objective: To evaluate the application's performance under a sustained load over an extended period.				
Test ID	**Description**	**Test Steps**	**Expected Result**	Pass/Fail
TC-PT-003	Endurance Testing	1. Identify workloads and user scenarios that represent a continuous load on the system.	The workloads should simulate continuous and sustained user activity.	Pass
		2. Set up cloud-based endurance testing tools or services to run tests for an extended duration, ensuring constant load on the application.	The endurance testing tools should be capable of running tests continuously without interruptions.	Fail
		3. Execute the endurance tests focusing on resource consumption, response time, and error rates over the extended duration.	The application's performance should remain stable, with no significant degradation observed over time.	Pass
		4. Analyze the performance data to identify any potential memory leaks or resource exhaustion issues that may occur over time.	Memory leaks or resource exhaustion issues should be identified, and recommendations for mitigating them should be provided.	Pass
		5. Verify if the application can sustain the load for the defined duration without any significant performance degradation or system instability.	The application should demonstrate stability and responsiveness over the extended testing period.	Pass
		6. Assess the application's ability to recover from any potential issues or resource exhaustion during the endurance tests.	The application should be able to recover gracefully from any issues encountered during the extended testing period.	Pass
		7. Check for any long-term memory growth or other performance trends that may impact the application's ability to handle sustained workloads over time.	The application's performance should not deteriorate over time, and memory usage should not experience significant long-term growth.	Pass

V. CONCLUSION

The need for structured Cloud-Based software testing to address the challenges identified through the literature review has been proposed. The new model streamlines the testing activities in the cloud environment and simplifies the testing. A case study implementation with the test cases designed for CI/CT testing, security testing, and performance testing using smoke testing and sanity testing methods are discussed. Each testing method used in this work provided test cases for various scenarios to ensure the maximum coverage of test conditions. All the test cases derived are executed, verified and reported as per the software requirements.

VI. FUTURE SCOPE

The cloud-based testing model can be automated to derive test cases and their execution so that the time and cost will be reduced. Also, increase the number of test conditions to widen the scope of testing to identify more bugs in the early stage to reduce the overall cost of testing.

REFERENCES

[1] Daryl Elfield, Mark Corns, and Priya Raju, **(**2020), “Software testing in the cloud”, https://assets.kpmg.com/content/dam/kpmg/uk/pdf/2020/11/software-testing-in-the-cloud.pdf.

[2] C. H. Kao, S. T. Liu and C. C. Lin, "Toward a Cloud Based Framework for Facilitating Software Development and Testing Tasks," 2014 IEEE/ACM 7th International Conference on

Utility and Cloud Computing, London, UK, 2014, pp. 491-492, doi: 10.1109/UCC.2014.66.

[3] Liu, Zhenyu & Chen, Mingang & Cai, Lizhi. (2014). A Novel Automated Software Test Technology with Cloud Technology. 712-716. 10.1109/UIC-ATC-ScalCom.2014.63.

[4] S. Ali and H. Li, "Moving Software Testing to the Cloud: An Adoption Assessment Model Based on Fuzzy Multi-Attribute Decision Making Algorithm," 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA), Tokyo, Japan, 2019, pp. 382-386, doi: 10.1109/IEA.2019.8714986.

[5] J. Chen, C. Wang, F. Liu and Y. Wang, "Research and Implementation of a Software Online Testing Platform Model Based on Cloud Computing," 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), Shanghai, China, 2017, pp. 87-93, doi: 10.1109/CBD.2017.23.

[6] Reshma D. Abhang, Prof. B. B. Gite, 2014, Testing Methods and Tools in a Cloud Computing Environment, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 03, Issue 11 (November 2014).

[7] S. Nachiyappan, S. Justus, Cloud Testing Tools and its Challenges: A Comparative Study, Procedia Computer Science, Volume 50, 2015, Pages 482-489, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.04.018.

[8] N. Gokilavani, B. Bharathi, Multi-Objective based test case selection and prioritization for distributed cloud environment, Microprocessors and Microsystems, Volume 82, 2021, 103964, ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2021.103964.

[9] Amira Ali, Huda Amin Maghawry, Nagwa Badr, 2022, "Performance testing as a service using cloud computing environment: A survey", Journal of Software: Evolution and Process, Volume 34, Issue 12, https://doi.org/10.1002/smr.2492.

[10] Krichen, M. (2023). How Artificial Intelligence Can Revolutionize Software Testing Techniques. In: Abraham, A., Bajaj, A., Gandhi, N., Madureira, A.M., Kahraman, C. (eds) Innovations in Bio-Inspired Computing and Applications. IBICA 2022. Lecture Notes in Networks and Systems, vol 649. Springer, Cham. https://doi.org/10.1007/978-3-031-27499-2_18.

[11] Muhammad Babar, Ata Rahman, Fahim Arif, 2017, "Cloud Computing Development Life Cycle Model (CCDLC)", FUTURE 5V, Springer, DOI: 10.1007/978-3-319-51207-5_19.

[12] Alshazly, A.A., ElNainay, M., El-Zoghabi, A.A., & Abougabal, M.S. (2020). A cloud software life cycle process (CSLCP) model. Ain Shams Engineering Journal.

[13] Thomas Hamilton, 2023, Smoke testing, https://www.guru99.com/smoke-testing.html

[14] Nazneen Ahmad, 2023, Sanity Testing Tutorial: A Comprehensive Guide With Examples And Best Practices https://www.lambdatest.com/learning-hub/sanity-testing.