Global Journal of Engineering and Technology Advances
eISSN: 2582-5003
Cross Ref DOI: 10.30574/gjeta
Journal homepage: https://gjeta.com/

(REVIEW ARTICLE)

Check for updates

Hybrid software testing model to improve software quality assurance

Mensah Yaw Agyei [1,*], Ogunlolu Abigail Ajoke [1] and Mensah Yaw Asa [2]

[1] *Department of Software Engineering, Babcock University, Ilishan Remo, Ogun State, Nigeria.*

[2] *Department of Information Technology, Valley View University Techiman Campus, Ghana.*

Global Journal of Engineering and Technology Advances, 2023, 17(01), 104–116

Publication history: Received on 30 August 2023; revised on 08 October 2023; accepted on 11 October 2023

Article DOI: https://doi.org/10.30574/gjeta.2023.17.1.0206

Abstract

Software testing is a very critical factor in determining the quality, robustness and reliability of applications. Testing methods have advanced to accommodate the various software development practices that are being churned out at frequent rates and representing complex systems. One such method is the hybrid software testing model, which combines the strengths of different testing techniques to achieve comprehensive testing coverage. This study provides an in-depth analysis of the hybrid software testing model, exploring its definition, benefits, challenges, and best practices. Furthermore, it discusses various testing techniques commonly used in hybrid testing and presents case studies showcasing the successful implementation of the hybrid model in real-world scenarios.

Keywords: Software testing; Hybrid; model; Manual Testing; Automated Testing

1. Introduction

In general, products meant for public consumption are expected to pass through quality assurance checks or testing, to ascertain their conformity with rules and regulations guiding them. Quality Assurance is a key part of the product design cycle. Any product(s) that do not pass quality checks are discarded as defective or further reviewed. The quality of testing would determine the quality of the finished product[1]. When comprehensive tests are not applied to products, the deployed product is usually laden with defects or bugs. This analogy also applies to software products. Software testing is an integral part of the software development life cycle (SDLC), aimed at identifying defects or bugs in software products or systems. The primary goal of testing is to ensure that the software meets the specified requirements, functions as intended, and performs reliably in various scenarios [2]. Traditionally, testing was conducted in a manual sequential manner, following the waterfall model of development. Here, testers make test cases for the codes, test the software and give the final report about that software. No automation tools or scripts are involved. On the other hand, automated testing involves is the use of software separate from the software being tested to control the execution of tests and the comparison of actual outcomes with predicted outcomes[3]. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or perform additional testing that would be difficult to do manually.

However, with the emergence of agile and DevOps practices, testing methodologies have undergone significant transformations. Several software-testing models have been developed to address the diverse challenges in software testing[4]. These models include the V-model, iterative model and agile testing methodologies like Scrum and Kanban. Each model has its strengths and limitations, and their suitability depends on the project requirements, time constraints, and team dynamics[5]. For example, it is advisable to use the V Model on small-to-medium-sized software projects where the requirements are clear without any ambiguity. Whereas agile-scrum testing is often used when projects have a strong emphasis on delivering software quickly and frequently to meet market demands[6]. Scrum's iterative and

* Corresponding author: Mensah Y. A

incremental approach, coupled with efficient and timely testing, helps reduce time-to-market and enables the team to gather valuable feedback early on [7].

As software systems become more complex, employing a single testing methodology may not be sufficient to ensure comprehensive testing coverage [8]. Hybrid software testing models have emerged as a viable approach to overcome the limitations of traditional testing models. The hybrid model combines multiple testing techniques to achieve broader test coverage, enhance defect identification, and optimize resource utilization [9]. It refers to the integration of multiple testing approaches and methodologies to create a comprehensive and adaptable testing approach. It combines the strengths of different testing techniques, such as manual testing, automated testing, functional testing, and non-functional testing, to achieve thorough testing coverage across various dimensions of software quality [7].

Automated testing can be fast, repeatable, and reliable, making it ideal for testing large amounts of code and catching regressions quickly. However, automated tests may not catch all the edge cases and complex scenarios that manual testing can detect. Manual testing, on the other hand, can provide a more nuanced and exploratory approach to testing, enabling testers to uncover unexpected bugs and usability issues [10]. Furthermore, manual testing can be time-consuming, labor-intensive, and prone to human error.

The challenge, therefore, is to find the optimal combination of both testing models that can effectively detect defects and improve the quality of the software, while minimizing costs and time to market. This requires a well-planned and coordinated testing strategy, as well as the use of appropriate testing tools and techniques.

To this end, this study intends to implement a testing model that incorporates the benefits of manual testing as well as the advantages of the automated testing and establishes a cohesion, which would ensure that software testing in today's organizational settings would be as efficient as well as effective.

2. Overview Software Testing

Software testing is a critical and important part of the software development life cycle, aimed at evaluating the quality, stability, and functioning of software programs. It is a systematic process of executing software components or systems to detect errors, failures, or any departure from planned behavior. The objective of software testing is to detect problems early in the development process, minimizing the cost and risk of application failures in production settings [2].

Software testing encompasses a broad variety of activities that involve planning, designing test cases, executing tests, analyzing results, and reporting defects. The primary objective of software testing is to ensure that the software satisfies the specified requirements, functions accurately, and performs optimally under various conditions. It involves both functional and non-functional aspects, including usability, performance, security, and compatibility [11].

Figure 1 Manual Testing Procedure [11]

Various testing methods and techniques are employed in software testing. These include; manual testing, where human testers execute tests and validate the software's behavior, and automated testing which utilizes tools and programs to automate test execution and verification [12]. Both methodologies have their advantages and are often used in combination to accomplish comprehensive testing coverage. This comprehensive use-case forms the basis of this study as shown in figure 1 and 2.

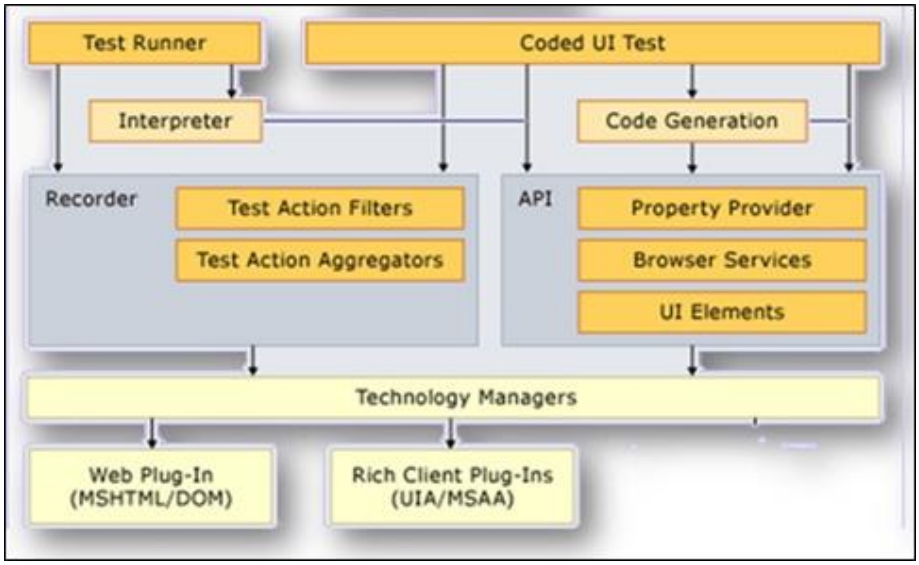

Figure 2 Automated Testing Setup/Flow [3]

Different forms of software testing pertain to specific objectives and phases of the SDLC. Some common forms of testing include unit testing, integration testing, system testing, acceptance testing, regression testing, and performance testing [13]. Each type serves a unique purpose, such as verifying the integrity of individual components, testing the integration between components, validating the entire system, or evaluating performance under various demands.

Software testing faces several challenges that can affect its effectiveness. These challenges include time and resource constraints, evolving requirements, complex software systems, and the need for skilled testers. Additionally, maintaining test environments, managing test data, and ensuring adequate test coverage can be demanding tasks. Overcoming these challenges requires effective planning, collaboration, and the use of appropriate testing methodologies and tools[14].

To overcome these challenges and ensure successful software testing, organizations follow best practices and guidelines. These include defining clear testing objectives, developing comprehensive test plans, prioritizing test cases based on risks, ensuring adequate test coverage, establishing robust defect reporting and tracking mechanisms, and promoting collaboration between development and testing teams.

2.1. Overview of Manual Software Testing

Manual software testing refers to the process of meticulously verifying and validating software applications by human testers. It involves the careful execution of test cases, interaction with the software's user interface, and observation of the software's behavior. Testers manually perform different types of testing, such as functional testing, regression testing, usability testing, exploratory testing, and ad-hoc testing, to ensure that the software meets the specified requirements [15] as shown in figure 2.3.

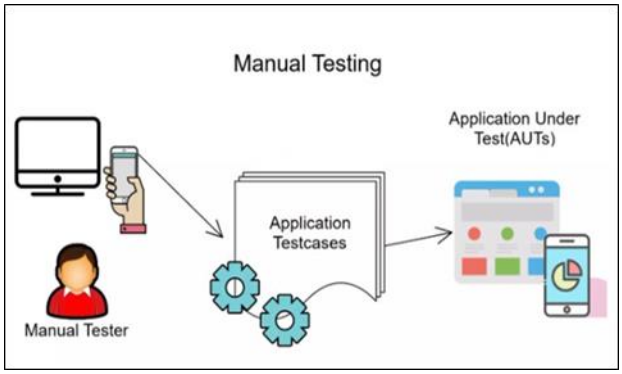

Figure 3 Manual Testing Process [16]

2.1.1. Advantages of Manual Software Testing

- **Early Bug Detection:** Manual testing allows testers to identify bugs and issues that might have been missed in automated testing. Human intuition and observation skills play a crucial role in detecting subtle defects, usability issues, and non-functional problems.
- **Exploratory Testing:** Manual testing enables exploratory testing, where testers have the freedom to explore the software, experiment with different scenarios, and uncover unforeseen defects. This flexible approach helps in improving the overall quality and user experience of the software.
- **User Perspective:** Manual testing provides insights into the user's perspective by allowing testers to interact directly with the software's user interface. Testers can evaluate the software's intuitiveness, ease of navigation, and overall user-friendliness, ensuring that the application meets the end-users' expectations[16].
- **Cost-Effectiveness:** Manual testing can be cost-effective, especially in the early stages of software development when frequent changes occur. Automated testing requires substantial effort to create and maintain test scripts, while manual testing allows testers to adapt quickly to evolving requirements and perform tests without significant upfront investment.
- **Flexibility and Adaptability:** Manual testing offers flexibility in designing and executing test cases. Testers can adjust their approach based on the specific requirements, risks, and changes in the software. Manual testing also adapts well to agile development methodologies, where frequent iterations and continuous feedback are vital as shown in figure 2.4[17].

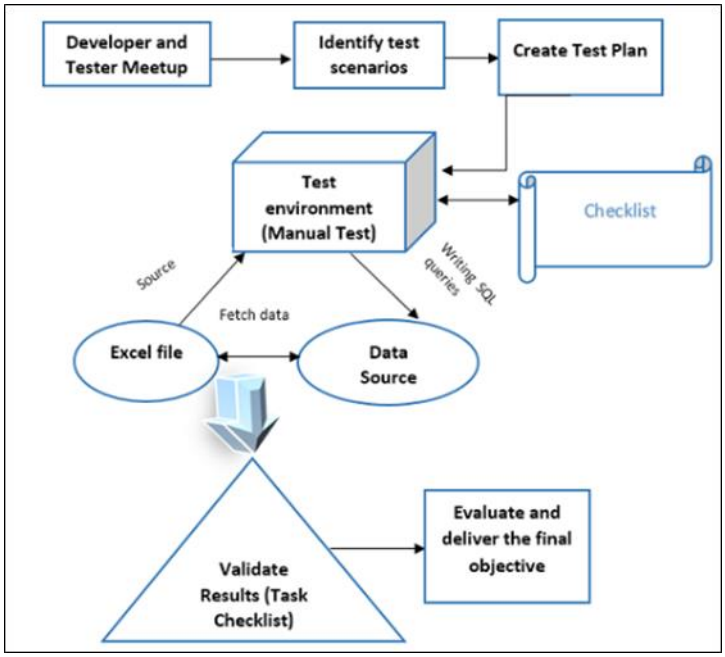

Figure 4 Manual Testing checklist [17]

2.1.2. Disadvantages of Manual Software Testing

- **Time and Resource Intensive:** Manual testing can be time-consuming, especially for large-scale projects with complex functionalities. It requires a dedicated team of skilled testers who invest significant effort in test case creation, execution, and result analysis[10].
- **Human Error:** Manual testing is susceptible to human error. Testers may overlook certain scenarios, misunderstand requirements, or introduce biases that can impact the quality of testing. Thorough training, documentation, and effective communication among the testing team can help mitigate these challenges.
- **Repetitive Tasks:** Manual testing often involves repetitive tasks, such as regression testing or retesting after bug fixes. These tasks can be monotonous, increasing the likelihood of errors. Testers need to maintain focus and concentration to ensure accurate and consistent results.
- **Scalability:** Manual testing becomes increasingly challenging as the size and complexity of the software application grow. Coordinating testing efforts, managing test data, and maintaining test coverage across various platforms and configurations can be demanding.

2.2. Overview of Automated Software Testing

Automated software testing has emerged as a vital component of the software development life cycle (SDLC), revolutionizing the way software applications are tested. It involves the use of specialized tools and scripts to execute test cases, validate functionality, and identify defects.

An automation tester writes the code for a test case or scenario to reduce the amount of time taken for manually testing the same scenario frequently. Even though to run the automation script and testing a scenario takes very less time, the development of the script is time consuming. Automation tools like Selenium, Eclipse, Maven, Git etc. are used in companies according to their needs.

Automated tests include;

- Background processes, file logging, and database entry are examples of hard-to-reach areas of the system.
- Frequently utilized features with a high risk of error: payment systems, registrations, and so on. Because crucial functioning tests take several minutes on average, automation assures quick errors.
- Load tests, which examine a system's functioning in the face of a huge number of requests.
- Template operations, such as data searches, the entry of multi-field forms, and the verification of their preservation.
- Validation messages: erroneous data should be filled in the fields, and the validation should be checked.
- End-to-end situations that take a long time.
- Data verification including precise mathematical computations, such as accounting or analytical processing.
- Verifying the accuracy of the data search. [3]

2.2.1. Advantages of Automated Software Testing

- **Improved Testing Efficiency**: Automated software testing significantly improves testing efficiency by executing test cases faster and more accurately than manual testing. The automation tools can quickly repeat test scenarios, eliminating the need for manual intervention, reducing human errors, and enabling faster feedback on software quality[18].
- **Increased Test Coverage**: Automated testing allows for broader test coverage by executing a large number of test cases across various scenarios, data sets, and configurations. It ensures that critical functionalities, edge cases, and integration points are thoroughly tested, reducing the risk of undetected defects[18].
- **Regression Testing and Continuous Integration**: Automated software testing is particularly beneficial for regression testing, which involves retesting previously validated functionalities after modifications or updates. By automating regression tests, developers can quickly identify any unintended side effects or regression issues, ensuring that existing functionality remains intact while new features are added. Automated tests also integrate seamlessly into continuous integration and delivery (CI/CD) pipelines, enabling frequent and efficient testing in agile development environments[18].
- **Increased Accuracy and Reliability**: Automated testing eliminates human error and subjectivity, resulting in more accurate and reliable test results. The tools precisely execute predefined test scripts, compare actual results with expected outcomes, and generate detailed reports. This consistency and objectivity enhance the reliability of the testing process and facilitate efficient debugging and issue resolution.

- **Resource Optimization**: Automated testing optimizes resources by reducing the need for a large manual testing workforce. Testers can focus on higher-value tasks such as test planning, test case design, and exploratory testing, while repetitive and time-consuming test execution are automated. This resource optimization leads to cost savings, improved productivity, and better utilization of human resources [18].

2.3. Review of Related Work

[19]conducted an experiemental hybrid testing model on a large-scale e-commerce application development project. The model combined manual testing techniques for critical workflows and user acceptance testing, while leveraging automated testing for regression testing, performance testing, and compatibility testing. This approach reduced time-to-market, improved software quality, and enhanced customer satisfaction. During this experiment, functional tests were carried in batches. With majority being performed by testing interns who only had the knowledge of manual testing. They worked with comprehensive acceptance criteria. The automated was performed using pre-written javascript codes that were saved to Postman libraries.

[20] adopted hybrid testing model to address functional and non-functional testing requirements. Manual testing techniques were employed for complex business logic scenarios, while automated testing was used for regression testing, security testing, and performance testing. The hybrid approach enabled thorough testing coverage and early defect identification, ensuring the reliability and security of the financial software.

In a mobile application development project, a hybrid testing model was implemented to validate the application's functionality, usability, and performance across multiple devices and platforms. Manual testing techniques were utilized for user interface testing and exploratory testing, while automated testing was employed for regression testing, device compatibility testing, and load testing. The hybrid model accelerated the testing process, improved test coverage, and enhanced the overall user experience of the mobile application.

3. The Real Time Hybrid Testing Model

After a thorough study of all, the various testing techniques it was discovered that their various disadvantages could limit the seamless progress of having a qualitative test. However, the hybrid model of testing takes into consideration the merits of manual and automated tests. In order words, it is designed to overcome all the demerits inherent in both manual and automated tests. The hybrid testing model is not designed out of nothing but works hand in hand with some specific tools. These tools include:

- **The Hardware:** every software testing needs a suitable hardware for functionality. The kind of hardware in use is determined by the scope and software type. For our research we would use a Laptop PC for connection to web applications and automated software testing tools. The PC has a minimum system requirement of 8GB of RAM and core i3 processor. In addition to this, a mobile smartphone is used. The smartphone is used for testing mobile applications functionality.
- **The Testing Software:** these tools include specific windows operating system, android systems, Test Complete (for web and desktop tests), Postman, Selenium drivers and cypress. Selenium is used to automate web-based application testing, while Test Complete is used for mobile applications automated tests. These are used alongside Test Complete to get a qualitative testing process.

3.1. System Design

The system design aims to achieve the following:

- **Test Coverage:** the research model provides comprehensive test coverage by combining the strengths of manual and automated testing techniques. Ensuring that critical functionalities, user scenarios, and edge cases are adequately tested.
- **Flexibility:** the model is designed into a flexible system design that allows for the seamless integration of manual and automated testing. And it also provides the ability to switch between testing modes and adapt to changing project requirements.
- **Scalability:** the model is scalable and can handle varying project sizes and complexities. The system model supports the testing of small-scale projects as well as large-scale enterprise applications such as ERPS.
- **Test Case Management:** a robust test case management system is implemented that allows for the creation, organization, and maintenance of test cases. It ensures that the system supports easy retrieval, modification, and execution of test cases.

- **Reporting and Analysis:** reporting and analysis capabilities are incorporated into the model design to provide insights into test results, defect tracking, and overall test progress. It enables stakeholders to make informed decisions based on comprehensive test reports.
- **Test Management Module:** This is a module that facilitates the management of test cases, test suites, and test execution. It allows the tester to create and organize test cases, assign priorities, and define test configurations.
- **Test Execution Engine:** The test execution engine executes test cases either manually or through automation tools. It provides a seamless interface for executing automated tests and capturing results.

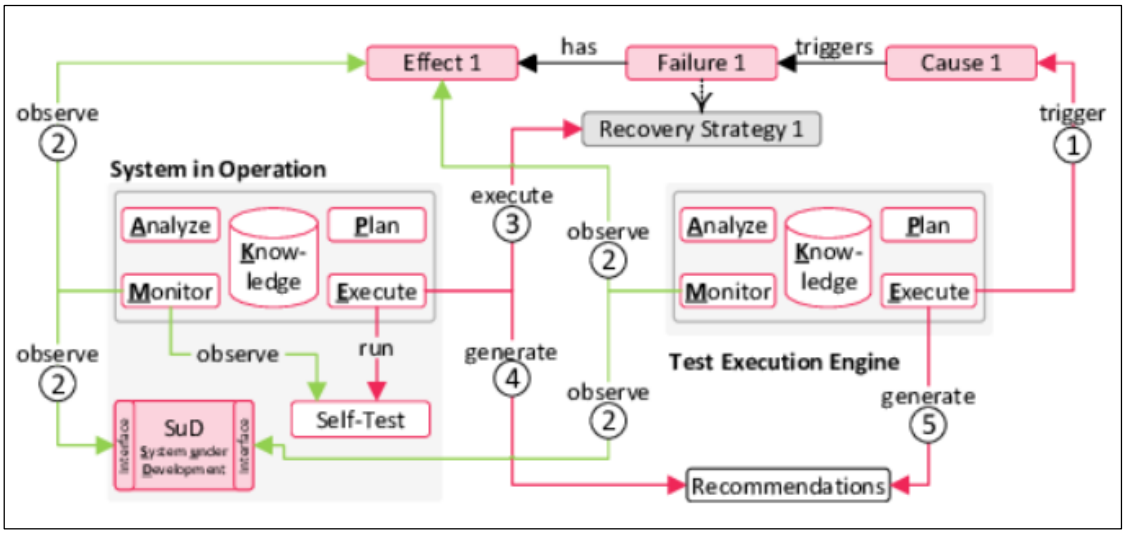

Figure 5 A Test Execution Engine operates

3.2. Hybrid Model Flowcharts

Hybrid model flowcharts are depicted to show various interactions between process stages.

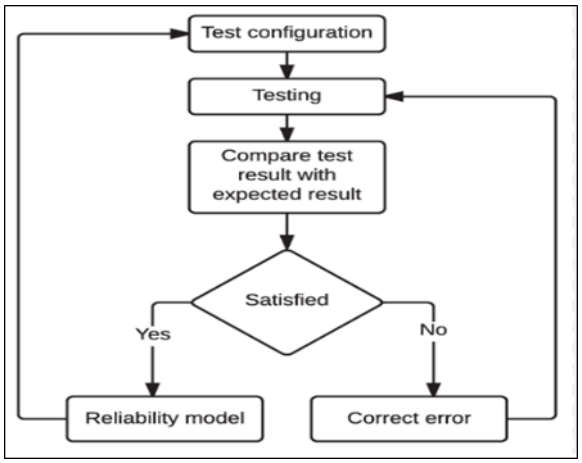

Figure 6 Workflow of Testing

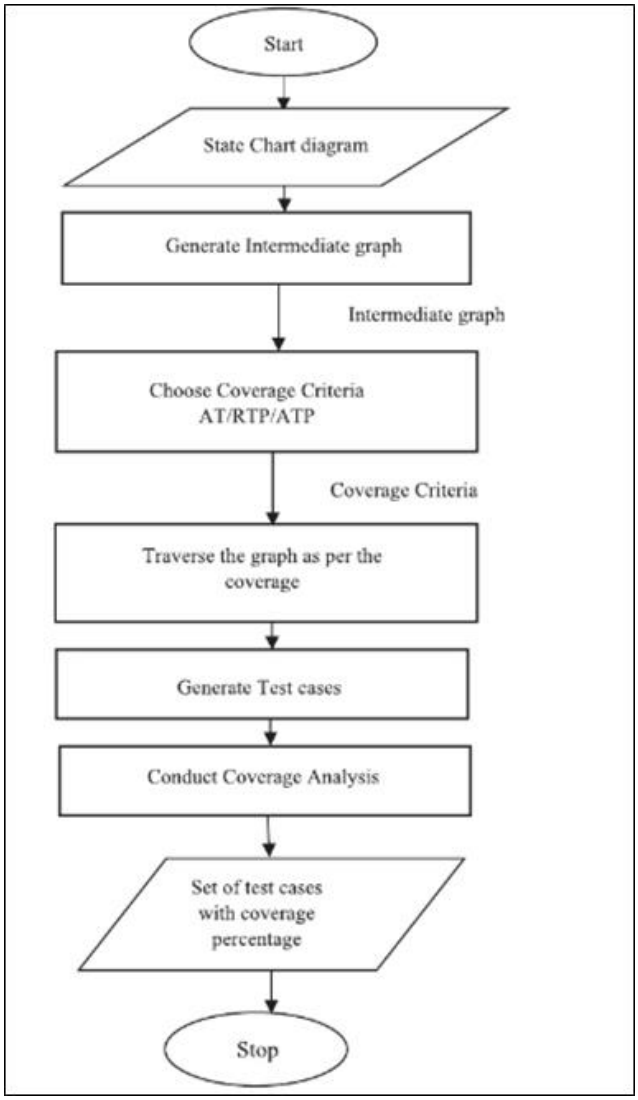

Figure 7 Flow for generating test cases

4. System Implementation

Table 1 and 2 shows implementation of the system. Test cases are built from the feature requirements of the software. A test case must contain key parameters for thoroughness. The following are important and can be captured manually:

- Test case ID: every test case has an identifier used to record its occurrence.
- Feature to be tested: this is based upon the requirements from the analysis phase of SDLC. Each feature usually has a negative test and a positive test.
- Test Case: This is the premise on which the testing is carried out. It mostly consists of actions to be performed. In this case, a manual approach is used for the first few tests in order to establish a familiarity with the software.
- Precondition: a precondition is a process/action that must have been performed before the actual testing must be carried out. A Pre-condition could be another test case in some scenarios.
- Expected Results: these are the required outcomes that must have been developed from the feature acceptance criteria given by business analyst or product owner.
- Test Status: this would determine the outcome of the tests. All test cases must be passed before there could be a sign-off for completion.

Table 1 Test case sample of the system

B	D	E	F	G
Test Case Id	Feature	Test Case	Precondition	Method/Steps
TC_01	Installation	Verify that user is able to successfully Install Application		1. Open Playstore application 2. Search for 3. Click on Install
TC_02	Launch App	Verify that user is able to successfully launch application	User has installed the application	1. Double click on the app icon
TC_03	Sign-In	Verify sign-in page details	1. User has launched the application 2. User is on the Sign-In page	1. Open 2. Click on **Log in**
TC_04	Sign-In	Verify that the **Phone Number** field is clickable	1. User has launched the application 2. User is on the Sign-In page	1. Click on the **Phone Number** field
TC_05	Sign-In	Verify that user is able to enter **Phone Number**	1. User has launched the application 2. User is on the Sign-In page	1. Click on the **Phone Number** field 2. Enter phone number

Table 2 Test Case result of the system

Method/Steps	Expected Results	Test Status - 16/02/23	Comments	Teat Status 30/03/23
1. Open Playstore application 2. Search for 3. Click on Install	The user should be able to download successfully	Pass		Pass
1. Double click on the app icon	1. App should be successfully launched 2. User should be able to see application's sign-in page	Pass		Pass
1. Open 2. Click on **Log in**	1. The user should be able to view the sign in page 2. It should contain the following; - **Phone number** field - **Pin** field - Forgot password - Sign in button	Pass		Pass
1. Click on the **Phone Number** field	The phone number field should be clickable	Pass		Pass

4.1. Hybrid Component Implementation

This involves performing a progressive overall testing of the system's objectives in order to make sure that all requirements and objectives are met at each stage of production. Here the automated tests from software tool is fused with the manual section to form an overall qualitative test.

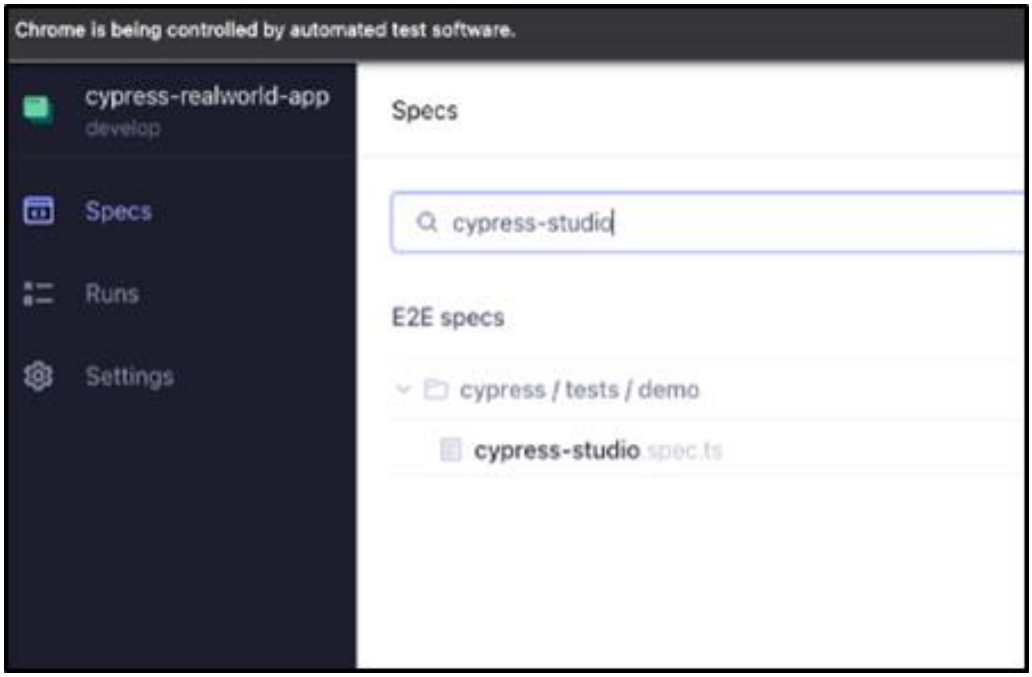

Figure 8 Cypress studio for running automated test scripts

```
/// <reference types="cypress" />
let textFunction;
describe('Login Tests', () => {

   it('Verify that user can successfully Login and and confirm the merchant store name', () => {
   cy.visit('/')
    cy.fixture("LoginPage").then((el) => {
      cy.get(el.LoginPage.email).should('be.visible').type(Cypress.env('username'))
      cy.get(el.LoginPage.password).should('be.visible').type(Cypress.env('password'))
      cy.get(el.LoginPage.password).should('have.attr', 'type', 'password')
      cy.get(el.LoginPage.loginBtn).should('be.visible').click()
      // verify that login is successful
     cy.xpath("//header/div[2]/button[1]/div[2]/h1[1]").should('have.text','noel-nolan')
   })
  })

  it('Verify that user can successfully Login confirm that the welcome message with the first name is shown', () => {
   cy.visit('/')
    cy.fixture("LoginPage").then((el) => {
      cy.get(el.LoginPage.email).should('be.visible').type(Cypress.env('username'))
      cy.get(el.LoginPage.password).should('be.visible').type(Cypress.env('password'))
      cy.get(el.LoginPage.password).should('have.attr', 'type', 'password')
      cy.get(el.LoginPage.loginBtn).should('be.visible').click()
      // verify that login is successful
      cy.xpath("//body/div[@id='root']/main[1]/div[2]/div[2]/div[1]/div[1]/div[1]/h1[1]").should('have.text','Hi Hokage,
   })
  })
})
```

Figure 9 Cypress Automated scripts

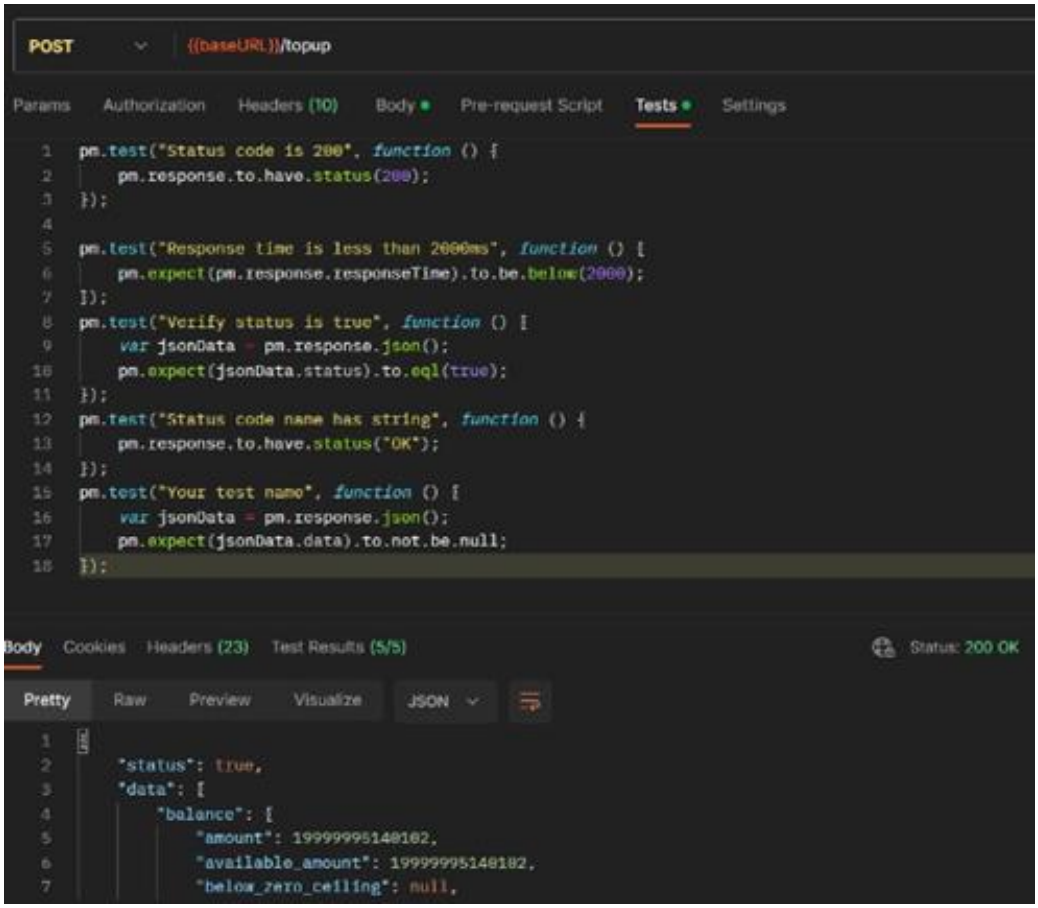

Figure 10 Automated output from testing

In the test case sheet, after manually testing by familiarizing with the features of the software, the tester proceeds to the Postman screen and views the output of scripts ran using automated testing. In this case, the output viewed was derived from Cypress automated tester. When testing has been concluded from the hybrid model, a comprehensive report is designed and assigned to the application feature or environment in which the tests were carried out. In this case, testing was conducted from a windows environment. This is seen from Table 3.

Table 3 System Implementation Report

Windows

Inventory Feature	Test Cases	Status	Comments
Select Store	Verify that the user is able to select a store from the stores available	Pass	
Sort Inventory	Verify that user is able to view all products, properties and situation	Pass	
Sort Inventory	Verify that user is able to sort inventory alphabetically	Pass	
Sort Inventory	Verify that user is able to sort inventory by categories	Fail	No categories present
Sort Inventory	Verify that user is sort inventory by order of entry; First in and Last in	Pass	
Sort Inventory	Verify that when a product has been sold, the product quantity is deducted from the total quantity of product in the store	Fail	The quantity remains the same before and after sale

5. Conclusion

The system design of a hybrid software testing model encompasses various components and considerations aimed at achieving comprehensive test coverage, flexibility, scalability, test case management, and reporting capabilities. By carefully designing and implementing such a system, organizations can harness the advantages of both manual and automated testing techniques, leading to improved software quality, faster time-to-market, and enhanced customer satisfaction.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Ann. Gravells, Principles and practices of quality assurance : a guide for internal and external quality assurers in the FE and skills sector, p. 189, Accessed: Oct. 06, 2023. [Online]. Available: https://books.google.com/books/about/Principles_and_Practices_of_Quality_Assu.html?id=nirFDAAAQBAJ

[2] L. Baresi and M. Pezzè, An introduction to software testing, Electron Notes Theor Comput Sci, vol. 148, no. 1 SPEC. ISS., pp. 89–111, 2006, doi: 10.1016/J.ENTCS.2005.12.014.

[3] Elfriede. Dustin, Thom. Garrett, and Bernie. Gauf, Implementing automated software testing : how to save time and lower costs while raising quality, p. 340, 2009.

[4] S. Al-Saqqa, S. Sawalha, and H. Abdelnabi, Agile software development: Methodologies and trends, International Journal of Interactive Mobile Technologies, vol. 14, no. 11, pp. 246–270, 2020, doi: 10.3991/IJIM.V14I11.13269.

[5] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, Agile software development methods: Review and analysis, VTT Publications, no. 478, pp. 3–107, 2002.

[6] Iterative vs. Incremental Development: A Comparison - Plutora. Accessed: Oct. 06, 2023. [Online]. Available: https://www.plutora.com/blog/iterative-incremental-development-comparison

[7] S. Tahvili and L. Hatvani, Basic software testing concepts, Artificial Intelligence Methods for Optimization of the Software Testing Process, pp. 7–33, 2022, doi: 10.1016/B978-0-32-391913-5.00013-0.

[8] F. Lonetti and E. Marchetti, Emerging Software Testing Technologies, Advances in Computers, vol. 108, pp. 91–143, Jan. 2018, doi: 10.1016/bs.adcom.2017.11.003.

[9] S. K. Shivakumar, Enterprise Web Application Testing, Architecting High Performing, Scalable and Available Enterprise Web Applications, pp. 179–198, 2015, doi: 10.1016/B978-0-12-802258-0.00006-8.

[10] Browser Stack, Manual Testing vs Automation Testing. Accessed: Oct. 06, 2023. [Online]. Available: https://www.browserstack.com/guide/manual-vs-automated-testing-differences

[11] G. D. Everett and Raymond. McLeod, Software testing : testing across the entire software development life cycle, p. 261, 2007, Accessed: Oct. 06, 2023. [Online]. Available: https://books.google.com/books/about/Software_Testing.html?id=dmG_Dc7QFWUC

[12] M. Khari, Empirical Evaluation of Automated Test Suite Generation and Optimization, Arab J Sci Eng, vol. 45, no. 4, pp. 2407–2423, Apr. 2020, doi: 10.1007/S13369-019-03996-3.

[13] G. J. Myers, The Art of Software Testing, Second edition. Glenford J. Myers, Software Testing, Verification and Reliability, vol. 15, no. 2, p. 256, Jun. 2005, Accessed: Oct. 06, 2023. [Online]. Available: https://www.oreilly.com/library/view/the-art-of/9780471469124/

[14] SOFTWARE TESTING - Dr. Sanjay Kumar Singh and Dr. Amarjeet Singh - Google Books. Accessed: Oct. 06, 2023. [Online]. Available: https://books.google.com.ng/books/about/SOFTWARE_TESTING.html?id=HdKwDwAAQBAJ&redir_esc=y

[15] R. Patton, Software Testing, Second Edition, p. 408, 2005.

[16] A Practitioner's Guide to Software Test Design: Lee Copeland: 9781580537919: Amazon.com: Books. Accessed: Oct. 06, 2023. [Online]. Available: https://www.amazon.com/Practitioners-Guide-Software-Test-Design/dp/158053791X

[17] Manual Testing Checklist |Professionalqa.com. Accessed: Oct. 06, 2023. [Online]. Available: https://professionalqa.com/manual-testing-checklist

[18] D. Graham and M. Fewster, Experiences of test automation: case studies of software test automation, p. 617, 2012.

[19] A. T. Aniwange, P. T. Nyishar, B. S. Afolabi, and A. O. Ejidokun, A Hybrid Software Test Automation for Educational Portals, NOVATEUR PUBLICATIONS INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY, vol. 8, pp. 2394–3696, Oct. 2021, doi: 10.17605/osf.io/4adp8.

[20] A. Jarzębowicz and P. Weichbroth, A Systematic Literature Review on Implementing Non-functional Requirements in Agile Software Development: Issues and Facilitating Practices, Lecture Notes in Business Information Processing, vol. 408, pp. 91–110, 2021, doi: 10.1007/978-3-030-67084-9_6.