ADDIS ABABA UNIVERSITY

COLLEGE OF NATURAL SCIENCES

An Integrated Automation Software Testing Framework To Support Behavioural-Driven Development Approach

Demiss Mammo

A Thesis Submitted to the Department of Computer Science in Partial Fulfilment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

October 2024

Addis Ababa University

College of Natural Sciences

Demiss Mammo

Advisor: Ayalew Belay (Ph.D.)

This is to certify that the thesis prepared by *Demiss Mammo*, titled: *An Integrated Software Test Automation Framework to Support Behavioural Driven Development Approach*. and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Name Signature Date

Advisor: _____

Examiner 1:_____

Examiner 2:_____

Abstract

Many software products fail due to poor quality caused by misunderstandings between producer and customer. To avoid these gaps behavioral driven development is the recommended approach. It encourages collaboration and focuses on delivering software that meets the end user's or customer's expectations and business objectives. Testing either manual or automated is an integral part of any software development approach. So, this is true for behavioral driven development but test automation with the appropriately integrated framework is a good choice for testing that is repetitive on multiple versions of the software product. This enables the tester to run a large number of tests in a short period of time and deliver high-quality software products. Still, existing framework needs enhancement in execution speed and integration to support behavioural driven development approach. This study presents an integrated automation software testing framework that supports and is suitable for a behavioural driven development approach to enhance the test execution speed and effectiveness of test automation tasks. The key components of the proposed framework are framework configuration, page object repository, behavioural driven development, version control, continuous integration, automated test scripts, application under test, test execution and reporting. The proposed framework is implemented using cucumber, cucumber step definition generator, visual studio code, JavaScript, Jenkins, cypress and cucumber test reporting tools. The framework can translate human-readable scenarios or behaviors/features of the software into executable test code through automatically generated step definitions, used to perform requirement analysis, retesting, regression testing, acceptance testing, automatic test execution and reporting by aligning automated tests with business requirements and user expectations. As the experiment result shows we improved the test execution time required to execute a single test case as compared with existing framework. Expert evaluation result also shows that 83% of the respondents have agreed on the suitability, integration, functionality and content of the framework. The proposed framework offers a valuable solution for organizations seeking to adopt BDD methodologies and improve their software testing automation processes. In the future, it is important to add artificial intelligence capability, API testing, performance testing and improve the framework.

Keywords: Software automation testing, software automation testing framework, test case, test script, Behavioural Driven Development.

Acknowledgment

First of all, I would like to forward my deepest love to the Almighty God for giving me the patience, wisdom, knowledge, and strength to complete this study successfully. Next, I would like to express my deepest gratitude and heartfelt thanks to my thesis advisor, Dr.Ayalew Belay, for his scholarly and kind assistance in reading and correcting this thesis. He replies with constructive and helpful comments immediately after I submit it. My heartfelt thanks also go to my dear wife "አቤ" who appreciates and motivates me in different ways and wisdom. Furthermore, I would like to thank INSA quality assurance team members, colleagues, and everybody who contributed to my research.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

API	Application Programming Interface
ATSG	Automation Test Script Generation
AUT	Application Under Test
BDD	Behavioural Driven Development
CD	Continuous Deployment
CI	Continuous Integration
DTP	Dynamin Testing Process
IEC	International Electro-Technical Commission
IEEE	Institution of Electrical and Electronic Engineering
IFSA	Integrated Framework for Automated Software Testing
ISO	International Standard Organization
ISTQB	International Software Testing Qualification Board
JSON	JavaScript Object Notation
POM	Page Object Model
QA	Quality Assurance
QC	Quality Control
STAF	Software Test Automation Framework
STLC	Software Testing Lifecycle
STP	Static Test Process
TC	Test Case
TM	Test Management
TMMI	Testing Maturity Model Integration
UAT	User Acceptance Testing
VS	Version Control

Chapter 1: Introduction

1.1 Background

The success of any software product is greatly dependent on its quality. Today, testing is seen as the best way to ensure the quality of any product. Quality testing can greatly reduce the cascading impact of rework of projects which has the capability of increasing the budgets and delaying the schedule. The need for testing is increasing as businesses face pressure to develop sophisticated applications in shorter timeframes [1].Testing is an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component [2]. Software testing is the first software quality assurance tool applied to control the software product's quality before its shipment or installation at the customer's premises [3]. Software testing is an activity that should be done throughout the whole development process [4]. It is an integral part of the software development life cycle that identifies the defects, flaws, or errors in the application [1]. Software testing depends more on what you want to test in software, and how you want to test. This will help you to decide the tools that you need to gather to perform the test. In deciding the nature of the testing, testers play a key role in an organization. Their contribution is vital in deciding the appropriate implementation for a given test [1]. As software systems evolve, software testing becomes a difficult and time-consuming task when performed manually. A new technique must be applied to improve the test process and reduce the test effort of a tester. Test automation is one of the techniques which can be applied [5].

In recent years, the adoption of Agile software development methodologies has gained significant popularity[6]. Among these methodologies, Behavioural Driven Development (BDD) [7, 8] has emerged as a promising approach to ensure collaboration between developers, testers, and stakeholders throughout the software development process. BDD emphasizes the use of natural language specifications, known as "scenarios" to drive the development and testing of software systems[8].To effectively implement BDD, an integrated software test automation framework is crucial[5, 9]. An effective automation testing framework can overcome the deficiencies of manual testing by designing integrated automation software testing [5]. Manual testing is tedious, error prone compared to what the computer can potentially do and it is not suitable when tests are repetitive and need huge test data therefore using automated testing is most effective and helps to improve test coverage [10]. Test coverage is an indication of the degree to which

the test item has been reached or “covered” by the test cases. The type of coverage that is relevant varies by the level of test. For example, unit test coverage is often expressed in terms of the percentage of code tested, and software or system test coverage can be a percentage of requirements tested. There is a need for specification of coverage or some other method for ensuring sufficiency of testing [2]. A more formal, rigorous automation testing framework will go far to reduce the risk that happens in the process of manual testing activities. A framework is considered to be a grouping of a set of procedures, directions, and standards that can be combined or followed as a whole to leverage the benefits provided by the framework. The automation testing framework is a wall that is carried to have an implementation environment for the test automation scripts [5].

1.2 Motivation

Recent trend shows every software development company needs to deliver a high-quality product on time with minimum cost and respond to their customer needs. BDD is among the most popular development which revolutionizes the software development process by promoting collaboration, improving test coverage, aligning development with business goals, and facilitating early bug detection. By adopting BDD practices, teams can deliver software that meets desired behaviors and full fills business requirements. So, to achieve the needs of software development companies and increase confidence in software products, the inclusion of automated testing with a standardized framework into the software development process is vital.

Automated testing can reduce maintenance costs, improve test accuracy, lower risks, promote reusability, increase testing team collaboration to understand software requirements, and reduce test script development, and execution time. Test automation framework can provide a standardized approach to testing, making it easier for individuals to acquire the necessary skills and contribute to the industry. By looking at the great benefits provided by the test automation framework that supports the BDD approach, the researchers are motivated to launch this research for the enhancement of software test automation activities and delivery of high-quality software products with minimum effort and time.

1.3 Statement of the Problem

Software projects fail due to poor quality. In most software product development projects, two quality gaps exist. The first one is the producer gap, which is the difference between what is specified and what is delivered, and the second one is the customer or end user gap, which is the difference between what the producer delivered and what the customer wanted[11]. BDD encourages collaboration and focuses on delivering software that meets the end user's or customer's expectations and business objectives[12]. Both the business and its technical team are on the same page to deliver the desired software that fulfils the entire business goal [13, 8]. Testing either manual or automated, is an integral part of any software development approach. So, this is true for BDD, but test automation with an appropriately integrated framework is a good choice for testing that is repetitive on multiple versions of the software product [5]. This enables the tester to run a large number of tests in a short period and deliver high-quality software products [14]. The framework that doesn't integrate the required components for test automation and aligns with the BDD principle or process makes it difficult to implement the BDD approach with test automation. Testers may develop test scripts against end user requirements. Therefore, there is a need for an integrated test automation framework that supports the BDD approach. By using these types of framework test management and user acceptance testing become well simplified[8].

There are various types of test automation frameworks each having its own architecture and may differ from the other based on their support for different automation key factors like reusability, supportability, test coverage, ease of test script maintenance, and so on. linear, module-based, data driven, keyword driven, and hybrid frameworks are known as the most popular frameworks [14, 5]. So far, different researchers have designed and implemented frameworks based on the existing types of frameworks to enhance automation testing activities. They include a web testing platform based on a hybrid automated testing framework[15], an automated software testing framework for web applications [16], an integrated software test process framework the case of selected Ethiopian software companies [17], integrated automation software testing framework [5] and automation testing framework as an efficient medium for testing web application. However, existing software test automation frameworks lack effectiveness for the BDD approach and integration of other important components required for test automation such as continuous integration (CI), version controlling, test automation scripting, test

execution and reporting components. Test execution speed was low because of the selected tools to implement the framework in existing frameworks. Due to the nature of these existing frameworks, managing test automation tasks with BDD is difficult. Thus, in this study, we propose an integrated software testing automation framework that supports the BDD approach to enhance test execution time and the delivery of high-quality software that satisfies end user needs. The framework ensures high suitability for BDD, functionality, maintainability, reusability and integration of required components for test automation.

1.4 Objectives

General Objective

The general objective of the study is to propose an integrated framework for software test automation that supports the BDD approach.

Specific objectives

To achieve the general objective of this study, the following specific objectives are aimed.

- ✓ Review literature and related works in the area of automation testing framework as well as concepts, principles and processes of the BDD approach.
- ✓ Design the proposed framework.
- ✓ Develop a prototype to implement the proposed framework.
- ✓ Evaluate the effectiveness and of the framework

1.5 Methods

We will use the following methods to achieve the general and specific objectives of the research work mentioned above.

- **Literature Review:** we will conduct a literature review related software development process, software testing, BDD process and principles, software test automation, and frameworks. This will provide a solid foundation and understanding of the current state of research and industry practices. Related literature from different sources (books, journals, internet, etc.) will be reviewed. After proper analysis of the current research gaps, we will propose a new automation framework to address those identified gaps.
- **Designing and Prototype Development Tools:** we will use Microsoft Visio 2019 to design the framework, visual studio code editor, Nodejs, JavaScript, cypress, Jenkins, GitHub, cucumber and step definition generator to develop the prototype

of the proposed framework. The details of each development tool will be discussed in the implementation section of this thesis report.

- **Evaluation:** we will evaluate the effectiveness of the framework through experiments and expert evaluation.

1.6 Scope and Limitation

The research aimed at designing and implementing an integrated software automation testing framework for functional, and acceptance testing of web applications based on BDD principle and practice. This research work doesn't consider mobile applications, API and other testing types such as non-functional testing like performance testing, security testing, and so on.

1.7 Application of the Result

This research is believed to produce great significance in the production of quality software applications with minimum cost and time. Some of the applications of this research work result are the following:

- Improve the quality of the software test automation process and increase testing stability or consistency.
- Accelerate customer feedback cycle and collaboration resulting in high quality for the feature or function delivery.
- Reduce maintenance cost, testing time, higher return on investment (ROI), increase test coverage and reusability.
- Enhance testing team collaboration and conduct continuous testing.
- Reduce software development risks.

1.8 Organization of the Rest of the Thesis

The rest of the thesis is organized as follows: Chapter two presents a review of literature related to software development methodology, BDD approach, software testing and testing framework. Chapter three discusses detailed related works that have been conducted on the software test automation framework. Chapter four shows the design of an integrated software testing automation framework and discusses components of the framework in detail. Chapter five presents the implementation of the prototype and evaluation result of the proposed framework, tools used to develop the prototype and the importance of each tool. Chapter six presents the conclusion, contribution of this study, and future works.

Chapter 2: Literature Review

2.1 Overview

This chapter deals with the review of important concepts concerning the research topic. It provides a detailed definition of software testing, the relationship between testing and software development models, the testing process, testing levels, testing techniques, test automation, and frameworks for test automation. It also provides an overview of the key concepts and principles related to BDD and software test automation, as well as their integration.

2.2 Software Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that the software product is defect-free or quality. It involves the execution of software/system components using manual or automated tools to evaluate one or more properties of interest [15]. It is an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspects of the system or component [2].

2.2.1 Testing process and components

The testing process is used to provide information on the quality of a software product, often comprised of several activities, grouped into one or more test sub-processes [18]. The fundamental test process comprises test planning and control, test analysis and design, test implementation and execution, evaluating exit criteria and reporting, and test closure activities[19]. As shown in Figure 2.1, the ISO/IEC 29119 model divides the test process into organization and project levels. The organization level aims to define the process for the creation and maintenance of organizational test specifications, such as organizational test policies and strategies.

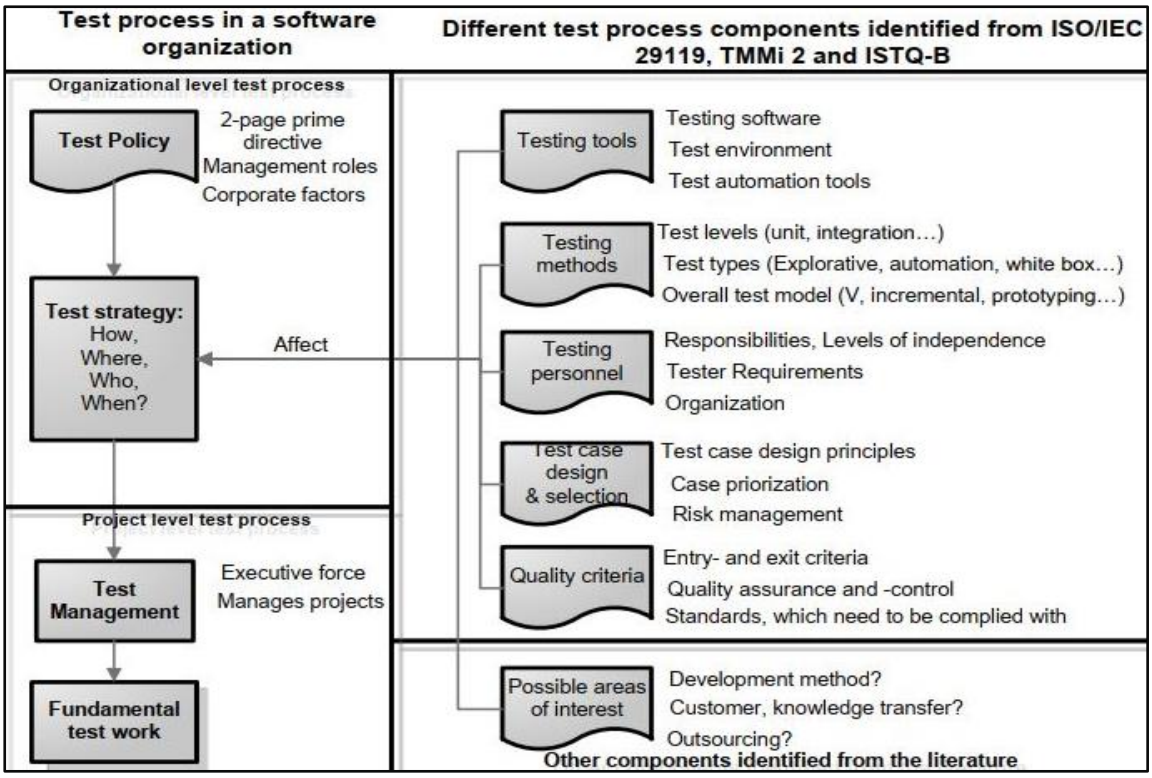

Figure 2.1 Test process and components [20]

Test policy: A high-level document describing the principles, approach, and major objectives of the organization regarding testing [19]. The test policy defines what testing is performed and what it is expected to achieve but does not detail how testing is to be performed. Test Policy can provide a framework for establishing, reviewing, and continually improving the organization's testing [21].

Test strategy: It is a document that expresses the generic requirements for the testing to be performed on all the projects run within an organization, providing details on how the testing is to be performed and should be aligned with organizational test policy [25]. As shown in Figure 2.4 [20], The organization defines a test strategy and uses testing policy, testing tools, testing methods, personnel, test case design, and selection as well as quality criteria as input.

On the other hand, the project level test process defines test management processes such as test planning, test monitoring and control as well as test completion.

Test management: It includes fundamental test work such as test planning, estimating, monitoring and control and test completion of test activities typically carried out by a test manager [19]. Figure 2.2 [22], illustrates the test management process and interaction between the components. Each component of the process has been described following the figure.

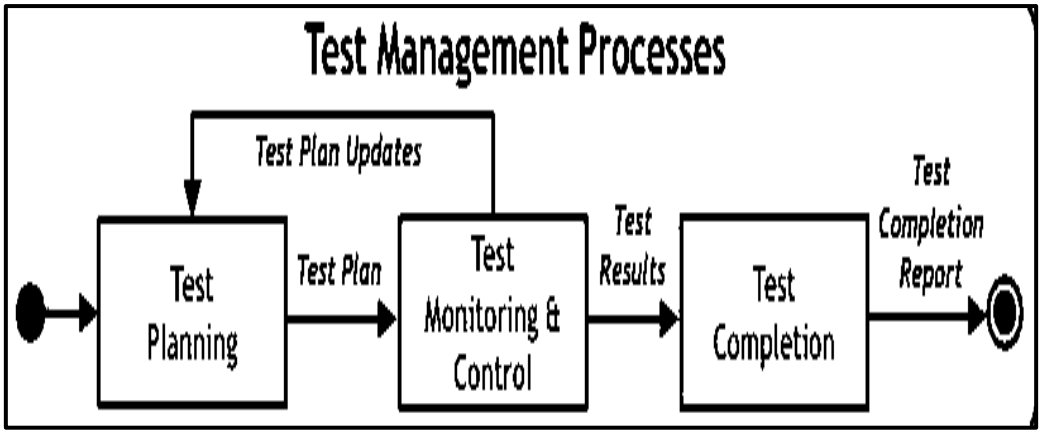

Figure 2.2 Test management process

Each component of the test management process will be discussed as follows:

Test planning: The test planning process is used to develop the test plan. Depending on where in the project this process is implemented this may be a project test plan or a test plan for a specific phase, such as a system test plan or a test plan for a specific type of testing like a performance test plan. The purpose of the test planning process is to develop, agree, record, and communicate to relevant stakeholders the scope and approach that will be taken to testing, enabling early identification of resources, environments, and other requirements of testing [22]

Test monitoring and control process: It is used to determine whether testing progresses according to the test plan and the organizational test specifications, such as the organizational test policy and strategy.

Test completion process - The test completion process is performed when the agreement has been obtained that the testing activities are complete. It will be performed to complete the testing carried out at a specific test phase (e.g., system testing) or test type (e.g., performance testing) and to complete the testing for a complete project. It ensures that useful test assets are made available for later use.

2.2.2 Types of Testing

A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on a specific test objective[24].

Dynamic testing: Under dynamic testing, code execution is required to check for the functional behavior of the software system, memory/CPU usage and overall performance of the system. The main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called validation testing. Dynamic testing executes the software and validates the output with the expected

outcome. Dynamic testing is performed at all levels of testing and it can be either black or white box testing [24]. Functional and non-function testing are categorized under dynamic testing and will be discussed as follows: -

Functional Testing: is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the functional requirements. Functional testing mainly involves black box testing and it is not concerned with the source code of the application [25]. The thoroughness of functional testing can be measured through functional coverage. Functional coverage is the extent to which some functionality has been exercised by tests and is expressed as a percentage of the type(s) of the element being covered. For example, using traceability between tests and functional requirements, the percentage of these requirements that are addressed by testing can be calculated, potentially identifying coverage gaps[23]

The following steps are involved in doing functional testing:

- Understand the functional requirements
- Identify test input or test data based on requirements
- Compute the expected outcomes with selected test input values
- Execute test cases
- Compare actual and computed expected results

Non-functional testing: evaluates characteristics of systems and software such as usability, performance efficiency, or security. Non-functional testing is the testing of "how well" the system behaves. The thoroughness of non-functional testing can be measured through non-functional coverage. Non-functional coverage is the extent to which some type of non-functional element has been exercised by tests and is expressed as a percentage of the type(s) of the element being covered [23].

Maintenance (change related testing): When changes are made to a system, either to correct a defect or because of new or changing functionality, testing should be done to confirm that the changes have corrected the defect or implemented the functionality correctly, and have not caused any unforeseen adverse consequences [23]. Confirmation and regression testing are the two main types of maintenance (change related)

- **Confirmation(re-test) testing:** After a defect is fixed, the software may be tested with all test cases that failed due to the defect, which should be re-executed on the

new software version. The software may also be tested with new tests to cover changes needed to fix the defect. At the very least, the steps to reproduce the failure(s) caused by the defect must be re-executed on the new software version. The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed.

- **Regression testing:** It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems. Changes may include changes to the environment, such as a new version of an operating system or database management system. Such unintended side-effects are called regressions. Regression testing involves running tests to detect such unintended side effects. Confirmation testing and regression testing are performed at all test levels. In regression testing, new tests are not designed, Instead, tests are selected, prioritized, and executed from the existing pool of tests [26].

Static Testing: Static testing is a type of software testing in which a software application is tested without code execution. Manual or automated reviews of code, requirement documents and document design are done to find the errors. It is also called non-execution testing or verification testing. Informal reviews, technical reviews, walkthroughs, inspections and static code reviews are techniques of static testing [24].

2.2.3 Levels of Testing

Testing is performed at different levels involving the complete system or parts of it throughout the life cycle of a software product. A software system goes through four stages of testing before it is deployed. These four stages are known as unit, integration, system, and acceptance level testing [26]. Each level of software testing will be discussed as follows: -

Unit testing: tests individual program units or components, such as modules, procedures, functions, methods, or classes, in isolation and usually performed by the developer in the developer environment [26]. Unit testing is independent testing which means no dependency is required to test a single component or module

Integration testing: It is also known as integration and testing (I&T),is a software development process in which program units are combined and tested as groups in multiple ways [27] . It is done after Unit testing. Every module involved in integration

testing should be unit testing before integration testing. While writing integration test cases, we don't focus on the functionality of the individual modules because individual modules should have been covered during unit testing. Here we have to focus mainly on the communication between the modules. Integration testing is needed dependent on another component or module to test a single component. There are four approaches to integration testing as big bang approach, top-down approach, bottom-up approach, and hybrid integration or sandwich testing.[28]. Each integration approach will be discussed as follows**.**

Big Bang Approach

Combining all the modules once and verifying the functionality after completion of individual module testing. In big bang integration testing, the individual modules are not integrated until all the modules are ready. Then they will run to check whether it is performing well. In this type of testing, some disadvantages might occur like, defects can be found at a later stage. It would be difficult to find out whether the defect arose in the interface or modules. As illustrated in Figure 2.3 [29], Once module A through module I completed and ready for testing then integrate them at once and test as a unit. If all of the modules in the unit are not completed, the integration process will not execute.

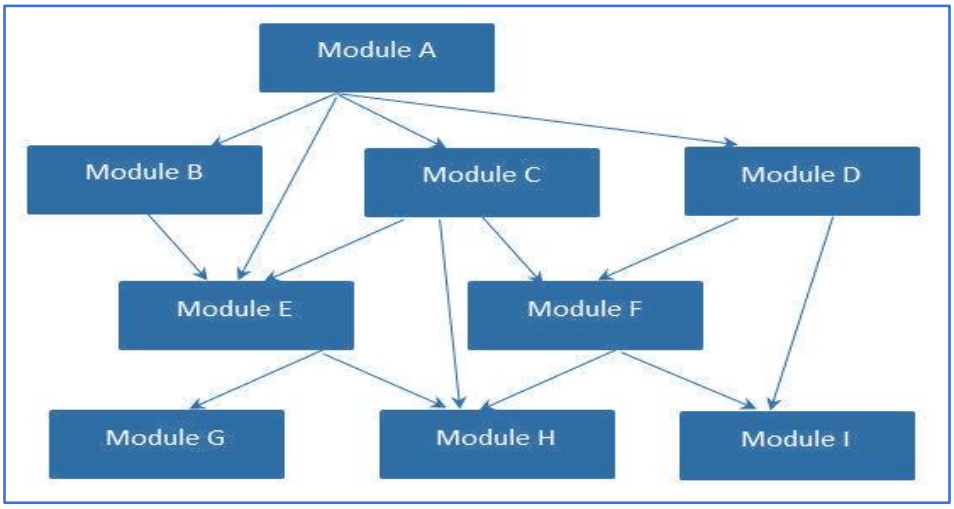

Figure 2.3 Big bang approach

Top-Down Approach

- In top-down integration testing, testing takes place from top to bottom. High-level modules are tested first then low-level modules and finally, the low-level modules to a high level to ensure the system is working as intended. In this type of testing,

Stubs are used as a temporary module if a module is not ready for integration testing. Figure 2.4 clearly states that Modules 1, 2, and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, Stubs are used to test the modules

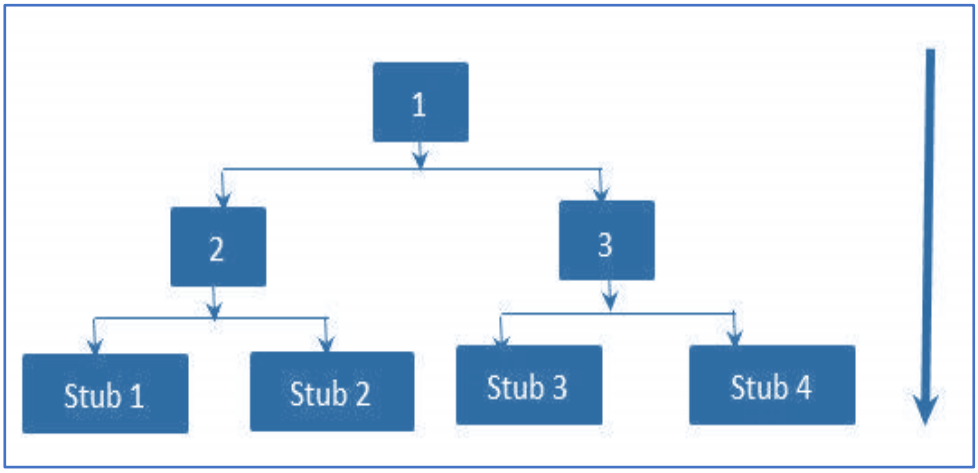

Figure 2.4 Top-down approach integration flow

Bottom-up approach

- It is a reciprocate of the top-down approach. In bottom-up integration testing, testing takes place from bottom to up. The lowest level modules are tested first and then high-level modules and finally integrating the high-level modules to a low level to ensure the system is working as intended. Drivers are used as a temporary module for integration testing. As shown in Figure 2.5 [30] the integration order will be: [4,2], [5,2], [6,3], [7,3], [2,1] and [3,1], then the whole system.

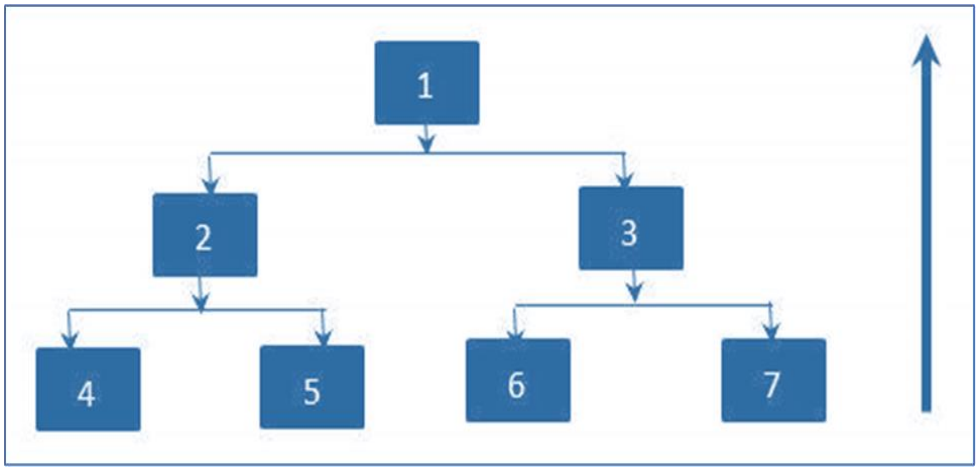

Figure 2.5 Bottom-up approach order of integration

Hybrid Integration Testing

Hybrid integration testing is also known as sandwich integration testing. It is the combination of both top-down and bottom-up integration testing. In hybrid integration testing, we exploit the advantages of top-down and bottom-up approaches [31].

System testing: System testing is the testing of an integrated system to verify that it meets specified requirements [19]. System testing focuses on the behavior and capabilities of a whole system or product, often considering the end-to-end tasks the system can perform and the non-functional behaviors it exhibits while performing those tasks. System testing should focus on the overall, end-to-end behavior of the system as a whole, both functional and non-functional, and typically carried out by independent testers who rely heavily on specifications [23].

Acceptance testing: Acceptance testing is formal testing concerning user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers, or other authorized entity to determine whether or not to accept the system[19]. Acceptance testing, like system testing, typically focuses on the behavior and capabilities of a whole system or product. Acceptance testing may produce information to assess the system's readiness for deployment and use by the customer (end-user). Defects may be found during acceptance testing, but finding defects is often not an objective, and finding a significant number of defects during acceptance testing may in some cases be considered a major project risk. Acceptance testing may also satisfy legal or regulatory requirements or standards [23].

2.2.4 Testing Techniques

White-box testing: This technique mainly focuses on the internal logic and structure of the code. White-box is done when the programmer has techniques full knowledge of the program structure[32]. It investigates the internal logic, code structure, and control flow of the application. For this testing technique, the tester must have strong programming skills [33].

Black box testing: It is a technique of software testing used to test the functionality of the application. This testing method is based on software requirements and specifications. It is a software testing technique where the internal workings of the item being tested are not known by the tester. It is also called specification-based testing and behavior testing. This testing handles both valid and invalid inputs according to the customer's requirement [33].

Grey box testing: This technique attempts, and generally succeeds, to combine the benefits of both black-box and white-box testing. Gray-box testing takes the straightforward approach of black-box testing but also employs some limited knowledge of the inner workings of the application. Therefore, a tester can verify both the output of the user interface and also the process that leads to that user interface output. Gray-box testing can be applied to most testing phases; however, it is mostly used in integration testing[32].

2.3 Testing Throughout Software Development Lifecycle Model

2.3.1 Waterfall Model

The waterfall model is a simple representation of the basic sequential model. It is possible to add verification tasks after each design activity, but the principle is that it is not possible to revisit an activity that is considered finished. Each activity ends with a milestone identifying whether the objectives of the activity have been reached, and deciding on the start of the next activity. Any going back would imply a re-execution of the completion milestone for this activity. The major drawback of a strict application of the waterfall cycle is that a defect generated in one phase, and detected in a later phase, should not imply correction of that defective initial phase. Another drawback is that testing starts very late in the project, with the associated risk of being skipped altogether due to timing constraints [34]. As shown in Figure 2.6 [35], once the development is completed, the testing phase starts, and in this phase, we test each unit or component and make sure the developed components are working as expected. All the testing activities are performed in this phase.

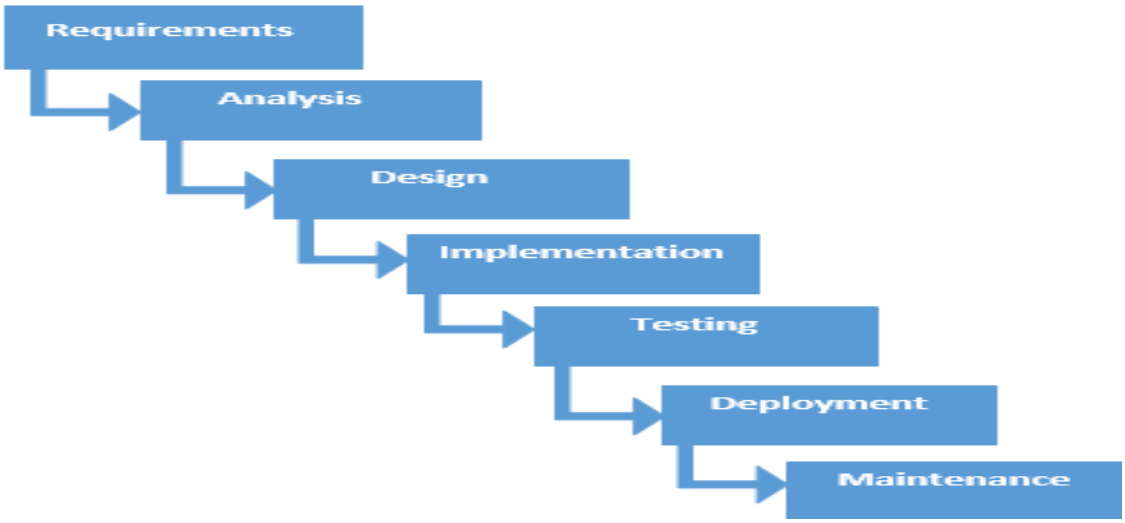

Figure 2.6 waterfall model

2.3.2 V-Model

V-model is a highly disciplined SDLC model in which there is a testing phase parallel to each development phase. The V_ model is an extension of the waterfall model in which testing is done on each stage parallel with development sequentially. It is known as the validation or verification model[36]. Figure 2.7 [37], illustrates the V-Model. With this model, the corresponding testing phase of the development phase is planned in parallel. So, there are verification phases on one side (left) of the 'V' and validation phases on the other side(right). The coding phase joins the two sides of the V-Model.

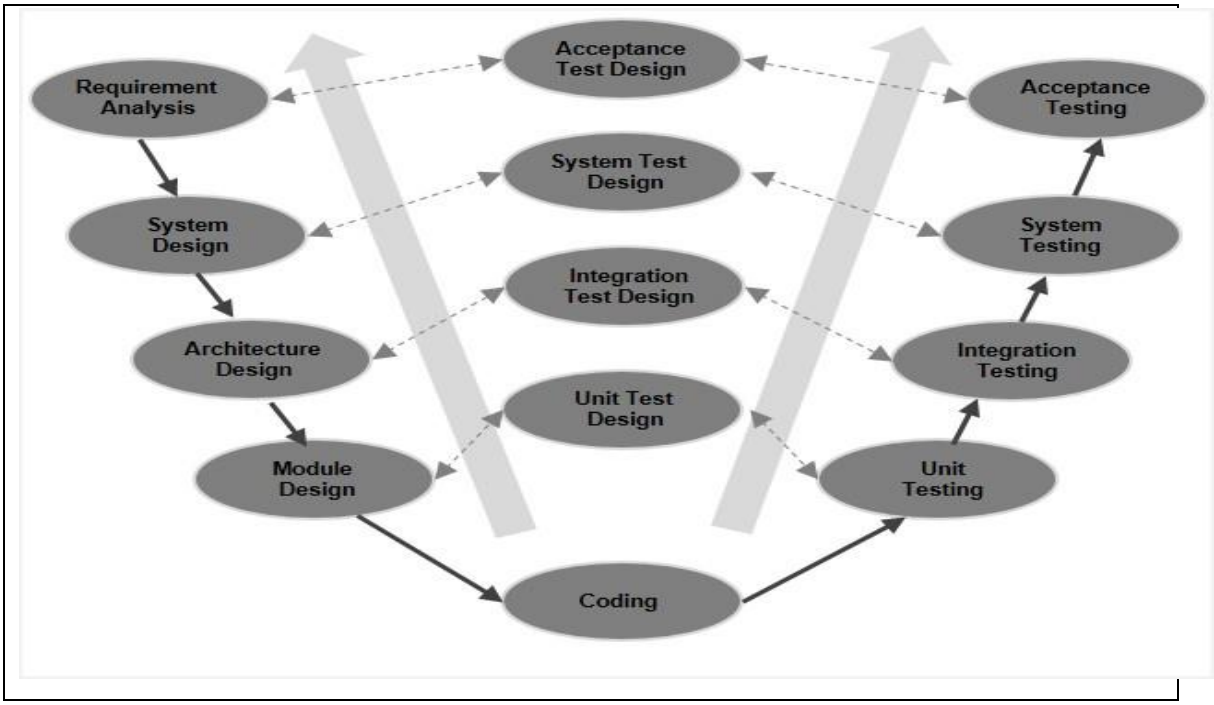

Figure 2.7 Testing with V-model

The main advantage of this model is that it associates a test activity with each design activity. It is to be noted that the activities are not specifically sequential but can overlap with higher and lower-level activities and the main drawback of this model is that shows the test activities as occurring after development, while test design activities should occur at the same time as the development [34]. Due to high rigidity adjusting scope or changing requirements is difficult and expensive and no early prototypes of the software are produced.

2.3.3 Iterative Model

Not all life cycles are sequential. There are also iterative or incremental life cycles where, instead of one large development timeline from beginning to end, we cycle through several smaller self-contained life cycle phases for the same project[38]. The iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iteration) and in smaller portions at a time (incremental) [39].

As illustrated in Figure 2.8 [40], with this incremental model, the whole requirement is divided into various builds or releases. During each iteration, the development module goes through the requirements, design, implementation, and testing phases. Each

subsequent release of the module adds a function to the previous release. The process continues till the complete system is ready as per the requirement.

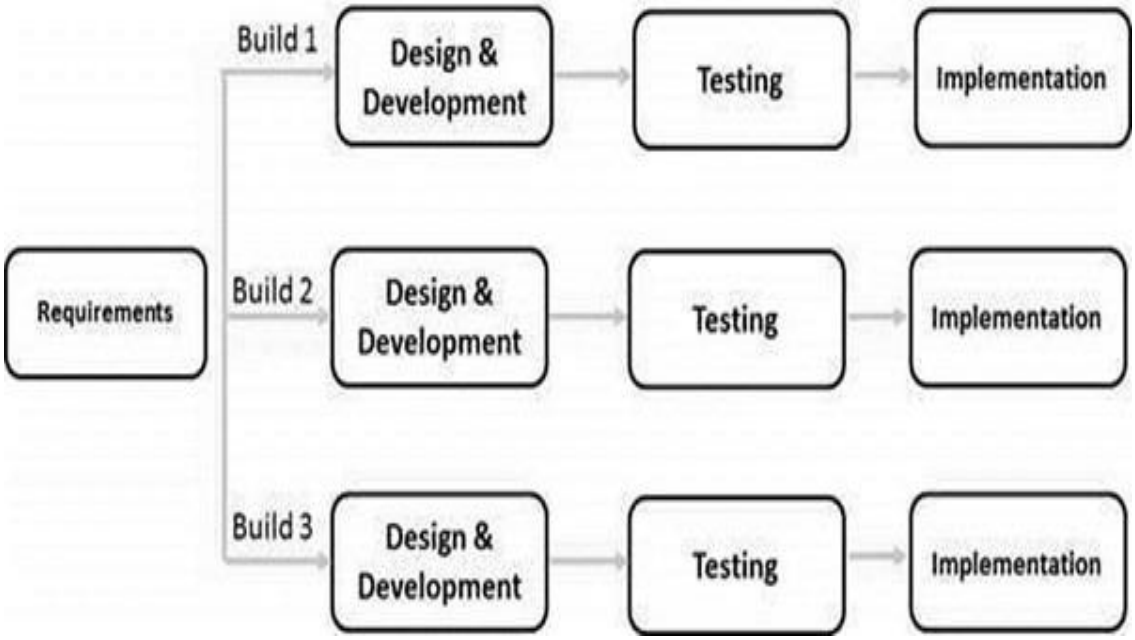

Figure 2.8 Testing with an iterative model

2.3.4 Agile Model

The “agile development models” are based on the principles exposed in the “Agile Manifesto” and are based on human and project realities, learning aspects, innovation, and continuous changes for better results. These methods are based on cross-functional teams (developers, testers, and customers) empowered to make design or implementation decisions, in opposition to long decision cycles functional compartmentalization and numerous and rapid iterations, with user feedback, instead of larger and less frequent deliveries [34].

Agile works on incremental methods. So, in the incremental method, the developing software has to be tested at each stage means after some development it has been tested with the customers and this type of testing is also done using automated acceptance testing. Agile is an approach to developing high-quality Software and fulfilling all the customer requirements with a self-organized team for delivering cost-effective Software within a time frame that also meets the user's requirement [41]. As illustrated in Figure 2.9 [42], Every iteration results in an integrated working product increment and is delivered for user acceptance testing. The customer feedback thus obtained would be an input to the next/subsequent Iterations.

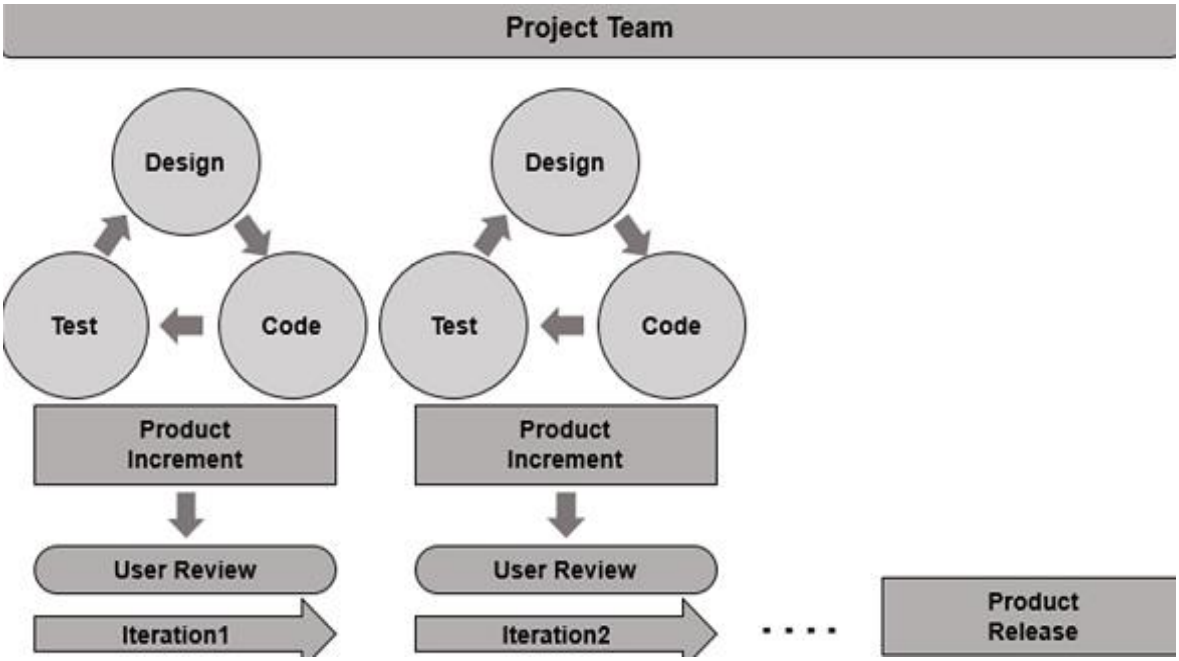

Figure 2.9 Testing in the agile (scrum)development

Agile removes the disadvantages of the waterfall model and v-model. It is an appropriate model for a project where requirements are changing and the project scope is not clear. The frequent involvement of customers at every step also increases the confidence and satisfaction of the customer in the end product and decreases the chances of a defect in the future. As client interaction is involved in every cycle so end product getting delivered after every cycle is according to the requirements. Agile testing also decreases the cost of the project as the working product is delivered after each cycle in increments so the chances of getting defects will become very low in the future.

2.4 Behaviour-Driven Development (BDD) Approach

2.4.1 BDD

It is a software development approach that allows the tester/business analyst to create test cases in simple text language (English) [43].

Most importantly, BDD provides a common language based on simple, structured sentences expressed in English (or in the native language of the stakeholders) usually using the Given/When/Then format that facilitates communication between project team members and business stakeholders. The concept of behaviour-driven development originated from the test-driven development (TDD) approach[8].

BDD helps teams focus their efforts on identifying, understanding, and building valuable features that matter to businesses, and it makes sure that these features are well-designed

and well-implemented. BDD isn't a software development methodology in its own right. It's not a replacement for Scrum, XP, Kanban, RUP, or whatever methodology rather it incorporates, builds on, and enhances ideas from many of these methodologies. The key to success in BDD lies with the execution of the acceptance tests which describe in an easily understood and easily definable manner a scenario that consists of defining the context, the executed actions and the expected response[44].

2.4.2 BDD Principles

Principle1: Focus on Features That Deliver Business Value

Uncertainty about requirements is a major challenge in many software projects, and heavy upfront specifications don't work particularly well when confronted with a shifting understanding of what features need to be delivered. A feature is a tangible, deliverable piece of functionality that helps the business to achieve its business goals. For example, suppose you work in a bank that's implementing an online banking solution. One of the business goals for this project might be "to attract more clients by providing a simple and convenient way for clients to manage their accounts." Some features that might help achieve this goal could be "Transfer funds between a client's accounts," "Transfer funds to another national account," or "Transfer funds to an overseas account." [8].

Principle2: Work Together to Specify Features or Behavior

A complex problem, like discovering ways to delight clients, is best solved by a cognitively diverse group of people that is given responsibility for solving the problem, self-organizes, and works together to solve it[8]. BDD is a highly collaborative practice, both between users and the development team, and within the team itself. Business analysts, developers, and testers work together with the end users to define and specify features, and team members draw ideas from their individual experience and know-how. This approach is highly efficient. In a more traditional approach, when business analysts simply relay their understanding of the users' requirements to the rest of the team, there is a high risk of misinterpretation and lost information [8].

Principle3: Mitigate Uncertainty in Software Development

A BDD team knows that they won't know everything upfront, no matter how long they spend writing specifications. As we discussed earlier, the biggest thing slowing developers down in a software project is understanding what they need to build. Rather than

attempting to lock down the specifications at the start of the project, BDD practitioners assume that the requirements, or more precisely, their understanding of the requirements, will evolve and change throughout the life of a project. They try to get early feedback from the users and stakeholders to ensure that they're on track, and change tack accordingly, instead of waiting until the end of the project to see if their assumptions about the business requirements were correct. Very often, the most effective way to see if users like a feature is to build it and show it to them as early as possible. With this in mind, experienced BDD teams prioritize the features that will deliver value, will improve their understanding of what features the users really need, and will help them understand how best to build and deliver these features [8].

Principle4: Illustrate Features/Behaviors with Concrete Examples

When a team practicing BDD decides to implement a feature, they work together with users and other stakeholders to define stories and scenarios of what users expect this feature to deliver. These examples use a common vocabulary and can be readily understood by both end users and members of the development team. They're usually expressed using the Given... When ... Then notation [8].

Principle5: Deliver Living Documentation

The reports produced by executable specifications aren't simply technical reports for developers but effectively become a form of product documentation for the whole team, expressed in a vocabulary familiar to users. This documentation is always up to date and requires little or no manual maintenance[45. It's automatically produced from the latest version of the application. Each application feature is described in readable terms and is illustrated by a few key examples. For web applications, this sort of living documentation often also includes screenshots of the application for each feature. Experienced teams organize this documentation so that it's easy to read and easy for everyone involved in the project to use. Developers can consult it to see how existing features work. Testers and business analysts can see how the features they specified have been implemented. Product owners and project managers can use summary views to judge the current state of the project, view progress, and decide what features can be released into production. Users can even use it to see what the application can do and how it works. Just as automated acceptance criteria provide great documentation for the whole team, low-level executable specifications also provide excellent technical documentation for

other developers. This documentation is always up to date, is cheap to maintain, contains working code samples, and expresses the intent behind each specification[8].

2.4.3 Different Phases of BDD

BDD mainly involves three phases or stages as illustrated in Figure 2.10 [46], [47], [48], [49]. Firstly, create a user story [50]for upcoming changes or enhancements to the system and talk about concrete examples [8] of the new functionality to explore, discover and agree on the details of what's expected to be done. The output of this phase is the agreed behavior of the system to be developed. Secondly, document those examples in a way that can be automated, and check for agreement. Lastly, implement the behavior described by each documented example, starting with an automated test to guide the development of the code[46].

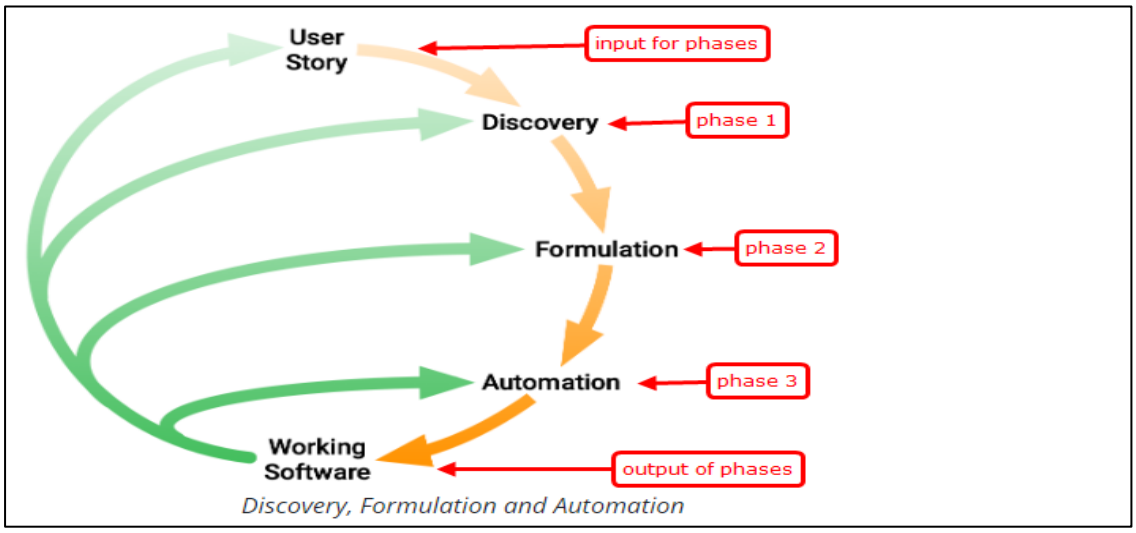

Figure 2.10 Different phases of BDD

Phase1: Discovery

This phase answers the question of what it could do. Once the user story [50] has been defined then a meeting happens between three peoples from different project teams. We call it the three amigos meeting. One person is the PO (Product Owner) who represents the stakeholders, one person is from the technical or development team, and one is from QA [47]. In this meeting, when the PO explains the functionality in the form of a user story, the Dev and QA then start asking questions and explore the functionality from all aspects [47]. The focus of this phase is on discovering the features that will add real value to the business [8]. The real goal is valuable, working software, and the fastest way to get there is through conversations between the people who are involved in imagining and

delivering that software [47]. The final output of this phase is the agreed behavior of the system to be developed [51]. The most common template to create a user story suggested in [52], [53] is as follows: *As a [type of user], I want to [action or goal], so that [reason or benefit]*.

For example**:** As the flying high sales manager I want travelers to earn frequent flyer points when they fly with us.

Phase 2: Formulation

This phase answers the question of what it should do. Then, all the discussed features [8] in the discovery phase should be formulated as structured documentation by using specification language called Gherkin's [54]. Gherkin uses a set of special keywords to give structure and meaning to executable specifications[46] This specification is an example of a behavior of the system. It also represents an acceptance criterion of the system. The primary purpose of this phase is to ensure that the acceptance criteria are confirmed and ready to be applied in real time. The team discusses the examples and the feedback is incorporated until there is agreement that the examples cover the feature's expected behavior. This ensures good test coverage[55]. Figure 2.11 [8], shows a sample formulation of features of the software that is to be developed.

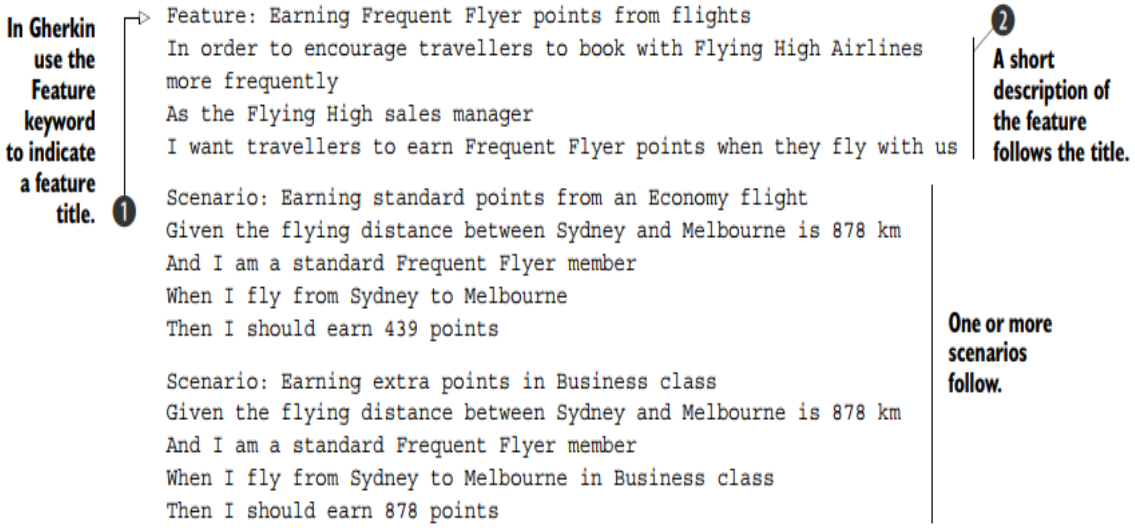

Figure 2.11 Sample formulated/documented executable and business readable examples or scenarios using Gherkins language with its special keyword

Phase 3: Automation

This is the final phase and answers the question of what it actually is. that means the formalized behavior of the system to develop turned into automated acceptance tests. The name automation phase automates the process of acceptance tests [60]. In this phase, developers write code to implement the desired behaviour based on the defined scenarios. Testers use the scenarios as test cases to verify that the software meets the specified requirements. Start by taking one feature at a time and automate it by connecting it to the system as a test. Now we develop the implementation code, using lower-level examples of the behavior of internal system components to guide us as required [46]. For example: Figure 2.13[8] shows the implementation of step definition that maps with steps in the feature file.

2.4.4 Benefits of BDD

In the previous sections, we examined what BDD looks like and discussed what it brings to the table. Now let's run through some of the key business benefits that an organization adopting BDD can expect in more detail.

Reduced waste*:* BDD is all about focusing the development effort on discovering and delivering the features that will provide business value, and avoiding those that don't. When a team builds a feature that's not aligned with the business goals underlying the project, the effort is wasted for the business. Similarly, when a team writes a feature that the business needs, but in a way that's not useful to the business, the team will need to rework the feature to fit the bill, resulting in more waste. BDD helps avoid this sort of wasted effort by helping teams focus on features that are aligned with business goals.

Reduced costs: The direct consequence of this reduced waste is to reduce costs. By focusing on building features with demonstrable business value (building the right software), and not wasting effort on features of little value, you can reduce the cost of delivering a viable product to your users. And by improving the quality of the application
code (building the software right), you reduce the number of bugs, and therefore the cost of fixing these bugs, as well as the cost associated with the delays these bugs would cause.

Easier and safer changes: BDD makes it considerably easier to change and extend your applications. Living documentation is generated from the executable specifications using

terms that stakeholders are familiar with. This makes it much easier for stakeholders to understand what the application actually does. The low-level executable specifications also act as technical documentation for developers, making it easier for them to understand the existing code base and to make their changes. Last, but certainly not least, BDD practices produce a comprehensive set of automated acceptance and unit tests, which reduces the risk of regressions caused by any new changes to the application.

Faster releases: These comprehensive automated tests also speed up the release cycle considerably. Testers are no longer required to carry out long manual testing sessions before each new release. Instead, they can use the automated acceptance tests as a starting point, and spend their time more productively and efficiently on exploratory tests and other nontrivial manual tests.

2.5 Automated Testing

2.5.1 What is Automated Testing?

Automation testing is the use of software to perform or support test activities, e.g., test management, test design, test execution, and reporting [19]

2.5.2 Automated Testing Process

For any automated tool implementation, the following are the phases/stages. Each one of the stages corresponds to a particular activity and each phase has a definite outcome. Automation testing is a fundamental part of the continuous development practice. Figure 2.12 [5]

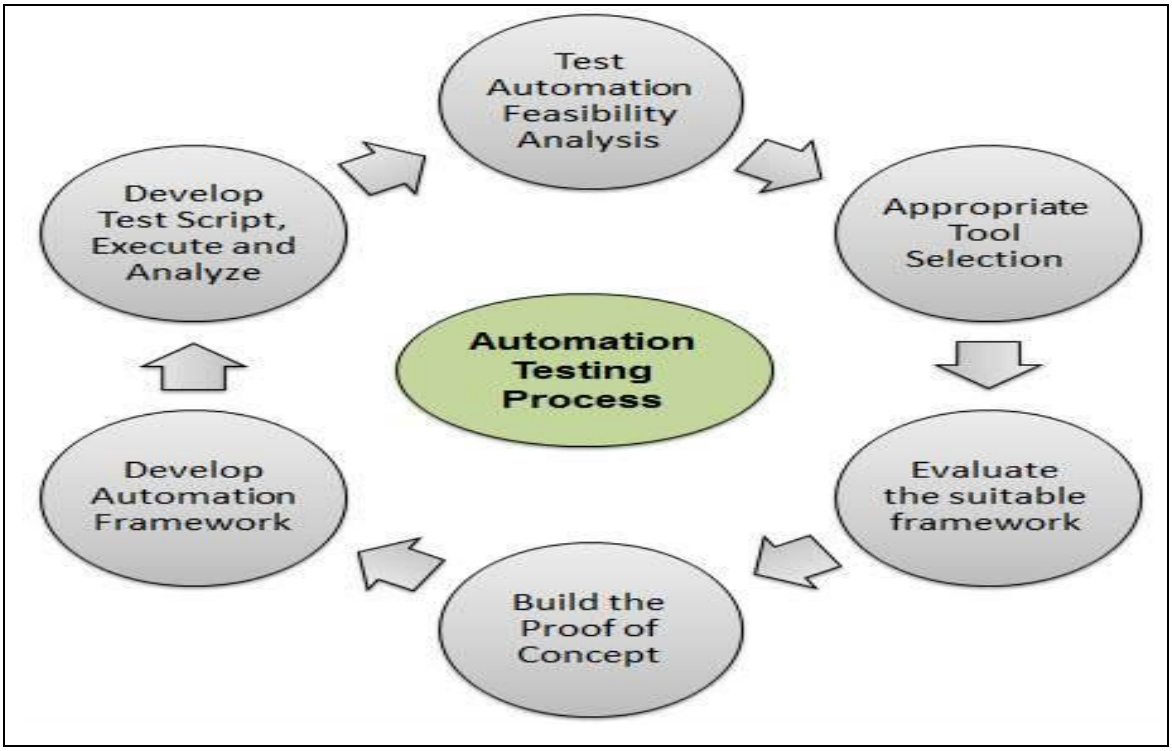

Figure 2.12 Test automation process

Test Automation Feasibility Analysis: The first step is to check if the application can be automated or not. Not all applications can be automated due to their limitations. Every aspect should be considered while analyzing the feasibility. Also, it is essential to perform a feasibility analysis on the manual test case pack that allows automation engineers to design the test scripts. In this particular phase, the following things should be taken care of without failure.

- ✓ Tester should select features of the applications to be automated and which not by considering the feature that is significant for the business, scenarios having large amounts of data, common functionalities throughout applications, the amount to which business components are reused, repetitive, and complexity amongst test cases.
- ✓ Determine which test types can be automated and how to automate them.
- ✓ Factors like cost, team size, and expertise should also be considered.

Appropriate Tool Selection: The next most important step is the selection of tools. To select tools, testers should consider the technology in which the application is built, its features and usage, the price of the tool, the programming language/platform/browser it supports, and the reporting mechanism. The tester should understand the product completely before starting the automation test. It is always a better idea to conduct a proof

of concept of the tool on AUT. Automation tools are used to automate certain sections of manual testing but not all

Evaluate The Suitable Framework: Upon selecting the tool, the next activity is to select a suitable framework type. There are various kinds of frameworks and each framework has its significance. Selecting a test automation framework is the first and foremost thing to do in the test strategy phase of the automation testing life cycle.

Build Proof of Concept: Proof of Concept (POC) is developed with an end-to-end scenario to evaluate if the tool can support the automation of the application. It is performed with an end-to-end scenario, which ensures that the major functionalities can be automated.

Develop Automation Framework: After building the POC, framework development is carried out, which is a crucial step for the success of any test automation project. The framework should be built after diligent analysis of the technology used by the application and also its key features.

Develop Test Script, Execute, and Analyse: Once script development is completed, the scripts are executed, results are analysed and defects are logged if any. The test scripts are usually version-controlled [56]

2.6 Test Automation Framework

2.6.1 What is Framework?

A framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful[57]

2.6.2 Test Automation Framework

A test automation framework is a comprehensive set of guidelines used to produce beneficial results of the automated testing activity and helps to execute software testing more efficiently. These guidelines may include common practices, assumptions for the desired outcome, a combination of testing tools and interfaces, test libraries, and coding standards [58]

2.6.3 Types of Test Automation Framework

There are various types of test automation frameworks each having its architecture and may differ from each other based on their support to different automation key factors like reusability, ease of maintenance, and so on [14]

Linear automation framework: With a linear test automation framework, also referred to as a record-and-playback framework, testers don't need to write code to create functions and the steps are written in sequential order the testers just record each step such as navigation, user input, or checkpoints, and then plays the script back automatically to conduct the test [9]

Modular based testing framework: This is a type of test automation framework in which the application under test is divided into separate functions, modules, or sections, each of which will be tested in isolation. A test script is created for each part and then combined to build larger tests in a hierarchical approach. These larger sets of tests will begin to represent various test cases. The key to achieving modularity is by decomposing the functionality and recombining the modules. The modular automation framework is also known as the functional decomposition framework[59].

Library Architecture Testing Framework: The library architecture framework for automated testing is based on the modular framework but has some additional benefits. Instead of dividing the application under test into the various scripts that need to be run, similar tasks within the scripts are identified and later grouped by function, so the application is ultimately broken down by common objectives. These functions are kept in a library which can be called upon by the test scripts whenever needed [9]

Data-Driven Framework: Using a data-driven framework separates the test data from script logic, meaning testers can store data externally. Very frequently, testers find themselves in a situation where they need to test the same feature or function of an application multiple times with different sets of data. In these instances, it's critical that the test data not be hard-coded in the script itself, which is what happens with a Linear or Modular-based testing framework. Setting up a data-driven test framework will allow the tester to store and pass the input/ output parameters to test scripts from an external data source, such as Excel spreadsheets, text files, CSV files, SQL tables, and so on. The test scripts are connected to the external data source and are told to read and populate the necessary data when needed[9]

Keyword-Driven Framework: This framework follows a similar approach to a data-driven framework, in such a way that the test data and script logic are also separated, but this approach takes it a step further. With this approach, keywords are also stored along with their associated objects in an external data source, making them independent from

the automation tool being used to execute the tests. Keywords are part of a script representing various actions being performed to test the software. These keywords can be labeled simply as ‘click,’ or ‘login,’ or with complex labels, like ‘click the link,’ or ‘verify link’, and the objects can be a ‘Submit Button,’ or ‘Login Username.’ For this approach to work properly, a shared object repository is needed to map objects to their associated actions [59]

Hybrid Test Automation Framework: As the name suggests, a hybrid framework is a combination of any of the previously mentioned frameworks set up to leverage the advantages of some and mitigate the weaknesses of others. Every application is different, and so should the processes used to test them. As more teams move to an agile model, setting a flexible framework for automated testing is crucial. A hybrid framework can be more easily adapted to get the best test results.

As Figure 2.13 [5], the hybrid testing framework is a combination of more than one of the above-mentioned frameworks. The best thing about such a setup is that it leverages the benefits of all kinds of associated frameworks.

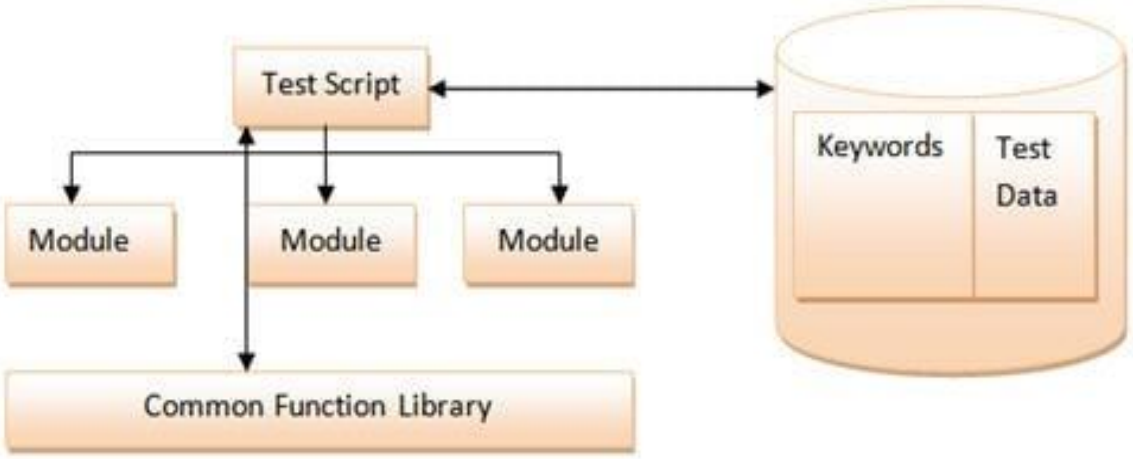

Figure 2.13 hybrid Test automation framework

Chapter 3: Related Work

3.1 Overview

Modern test automation framework is widely studied by different researchers to improve the problem of inefficient software test automation processes or activities. In this study, we present a review of some of the related work on software automation testing frameworks for web applications based on approach, domain, types of frameworks, automation tools they used to implement the framework and evaluation. It will discuss the features, strengths and limitations of each framework, focusing on their suitability for integrating BDD practices into the software testing process.

3.2 Data Driven Automated Testing Framework

Chandraprabha et.al[60] Proposed data driven testing framework using selenium web driver. In this research work the author design the test automation framework using selenium as automation tool, java programming language and TestNG as unit testing framework. The main focus of this work is separating test data from test script or storing it in some other external file. In this study reusability and maintainability of the framework is enhanced through the use of test data for different test scripts and modification of test data without affecting test scripts. However, the author doesn't consider BDD development approach, integration of CI, and version controlling systems. Selenium web driver by itself need more process to execute a test script that means communication between test script and AUT is via browser driver not direct execution on AUT so, this process reduces the performance of the framework.

Sonu Lamba et al [61] proposed an automated data driven continuous testing framework and they implemented the framework using selenium, TestNG, java and excel sheet. They implemented a method to retrieve test data from excel sheet and execute test cases. They generate test report but the test reporting mechanism was very poor and hard to understand by nontechnical stack holders they also missed integration of continuous integration and version controlling system.

3.3 Key Word Driven Automated Testing Framework

Arya K and Verma H [62] proposed a keyword driven automated testing framework for web application. The authors use an Excel sheet to store keywords and Selenium web Driver API[63] to import those keywords into the framework for executing test cases. As indicated in the experiment results the test report is not good and easy for non-technical users, the framework also does not support the BDD approach. The missed integration of version controlling and continuous integration features.

Milad *et al.* [16] proposed an automated software testing framework for web applications. Automating any test case using the proposed framework follows two major steps. The first step is to generate the test case steps sheet. The second step is to run the proposed framework on this generated test case steps sheet to generate their corresponding test script. The proposed framework can propose data values for each input HTML tag from a JSON file. During test execution no need to run the test manually. The absence of bug tracking, TM, CI, and version controlling integration is a major gap in this research work.

3.4 Hybrid Automated Testing Framework

Zhe et al. [15] proposed a web testing platform based on a hybrid automated testing framework that combines the technical advantages of keyword-driven and data-driven testing frameworks. The researcher adopts a hierarchical design pattern to separate test data from the test script, object library, and function library. The separation of the object library and function library can support the free and flexible combination of test scripts on demand, reduce the scale of test cases, and lower the maintenance cost of automated testing work. The focus of the proposed framework was to test a web application using keywords and data. It takes time to organize the keywords of each web element for test execution.

Tyagi et al [64]proposed the development of a reusable hybrid test automation framework for web-based scrum projects. They develop the framework by combining the features of both data driven and keyword driven frameworks using the Selenium 2.0 tool. The primary focus of the framework was to ensure reusability and maintainability of test scripts to speed up the testing process of web applications. They used of page object model to design the structure of the framework and implemented it with three layered architectures the first layer contains Application Under Test (AUT) that could be run on any browser such as Internet Explorer, Google Chrome, or Firefox, layer. The second layer contains an actual

hybrid test automation framework that is built using the Selenium web driver. They integrated good test reporting features but the framework does not support the BDD approach and does not integrate continuous and version controlling features.

3.5 Other Automated Testing Framework

Xianjie et al. [65] proposed the design and implementation of the bank financial automation testing framework based on Quick Test Professional (QTP). This work mainly targets the core business of the bank, credit, and online banking to test the function of the three major operations. The framework was developed based on QTP and mainly for regression testing of software, which integrates techniques including object recognition, data-driven, keyword-driven technology, and checkpoint technology, to proceed with business-level testing. The framework includes five parts: test driver layer, data layer, abstraction layer, object layer, and operate layer. This work only uses QTP to design a testing framework that is commercial and hard to scale up the framework. The proposed work was designed for banking applications only and ignored other types of software applications.

Md Nurul *et. al*[66] proposed a framework for the automation of cloud-application testing using selenium. The author of this research work designs the framework by using Java, Eclipse IDE, selenium web driver, page object design pattern, TestNg, and browser driver to conduct automation testing of cloud-based applications. By using this framework, the tester can conduct data driven testing but continuing testing is impossible due to the absence of CI, version controlling, and another important tools that are required for automation testing

Mekuannet Tilahun [5] proposed an integrated automation software testing framework. The objective of this work was to design and develop an integrated automation software testing framework to automate the design, execution, and result in an analysis of test cases by incorporating and integrating programming scripts, testing tools, source code management tools, automated build, and continuous integration (CI) servers. It takes input data from JSON files executes the test cases, and creates test reports. The proposed framework needs a third-party browser driver to execute the application under test and takes more time to complete test execution.

Shimelis Tamiru [17] proposed an integrated software test process framework: in the case of selected Ethiopian software companies. The author tries to show the manual way of the testing process, activities, and challenges, review different literature related to software

automation testing and propose a test improvement framework for the resource-constrained environment. In this research work, the author only shows the manual ways of software testing processes and techniques.

Gojare et al [67] proposed an analysis and design of selenium web driver automation testing framework. They implemented the framework using the Selenium WebDriver tool Object repository, input file, utility, test suite, and customized reports are the main components. They executed a test suite of 250 test cases on a student information system web application and compared the result with other traditional frameworks in terms of pass or failure rate, number of test cases per day, execution time, and maintenance cost. As the result shows the test efficiency using the framework was highly improved but the framework does not support the BDD approach and lacks features like integration of version controlling and continuous integration.

3.6 Summary

In this Chapter, the previous studies related to software test automation frameworks were reviewed and discussed. The authors in [60, 61] implement data driven framework using Selenium webDriver and Excel sheet to enhance automation testing through the reusability of testing codes but they did not consider the BDD approach, version controlling, reporting tool and continuous integration features. Throughout the review of the related works, we have seen almost all authors of previous research works implement the framework using previously known types of test automation framework and they missed supporting and integrating of BDD approach, version controlling, good reporting that suitable for all stakeholder and continuous integration features of the framework. Executing one test case using the existing framework takes a long time. Therefore, to improve the capability of the framework we propose an integrated software testing automation framework that supports the BDD approach, version control, good reporting mechanism that suits all stakeholders, continuous integration feature of a framework and improving test execution speed.

Chapter 4: Design of Proposed Framework

4.1 Overview

The overall objective of this chapter is to present the design of the proposed framework and the components that constitute the framework. We will show the interaction between each component. Following the design of the framework, a detailed description of each component and their interaction is presented.

4.2 Description of Components of the Framework

The proposed integrated test automation framework has nine components as shown in Figure 4.1. The components are: Framework configuration, BDD component, automated test script, page object model, version controlling, continuous integration, test execution, application under test and test reporting.

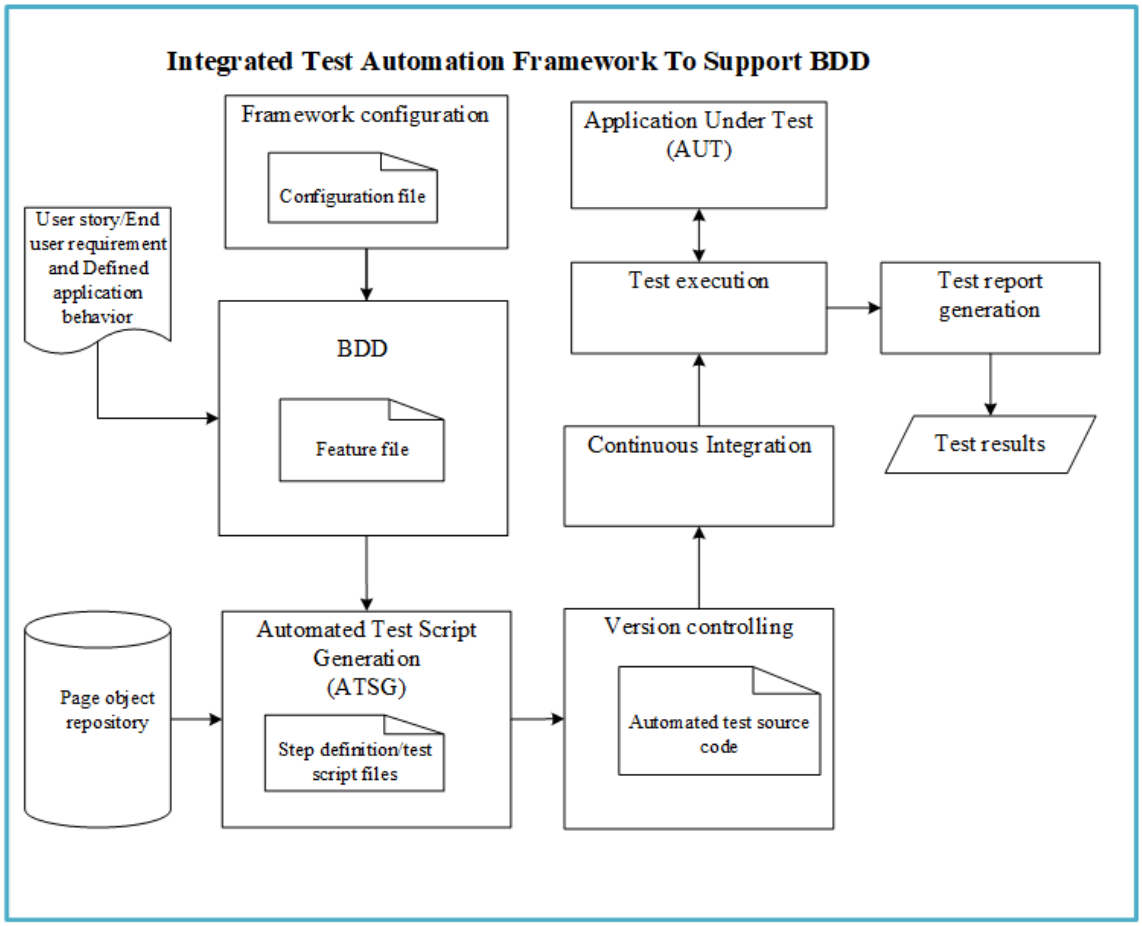

Figure 4.1 Integrated test automation framework

Framework Configuration: This component keeps some initial settings and specifies some key values that can be used throughout the test framework and will guide the behavior of the framework based on their default or altered values. The configuration depends on the tools used to implement the proposed framework. The files stored configuration settings such as tools integration, required dependencies, URL of AUT, browsers that used run test, number of tries, time out duration, file/folder to be included or excluded from test execution, and other related settings are among the common configuration of the framework. They allow testers to easily modify and manage the test automation settings.

BDD Component: This component is used to formulate feature file or specify the requirement based on the behavior that comes from the end user. The feature file consists of a feature description, scenarios, and test data. Scenarios represent specific instances of behavior or functionality to be tested. Test data is used to run tests with different input data to check if the tested software checks incoming values in a proper way. For example, if the application has a registration form, the tester wants to submit different combinations of user names, passwords, and emails to check the form behavior and test the form thoroughly, testers can pass both valid and invalid values to it.

Automated Test Scripts: This component implements feature files using programming language and automation testing tools. Once the requirements have been formulated in a structured way, the next step is transforming these created feature files into an automation test script in the form of a step definition file. Step definitions are created to map the steps in the feature files to the corresponding actual automation testing code implementation in the automation framework. These step definitions define the actions to be performed on software under test and the expected outcomes. When the BDD automated tests are executed, the automation framework reads the step definition file and executes the corresponding test scripts for each step in the feature file. A test script is a set of instructions written using a script /programming language that is performed on AUT to verify that the system performs as expected.

Figure 4.2, shows the process of generating automated test scripts in the form of a step definition file to execute the steps described in the feature file.

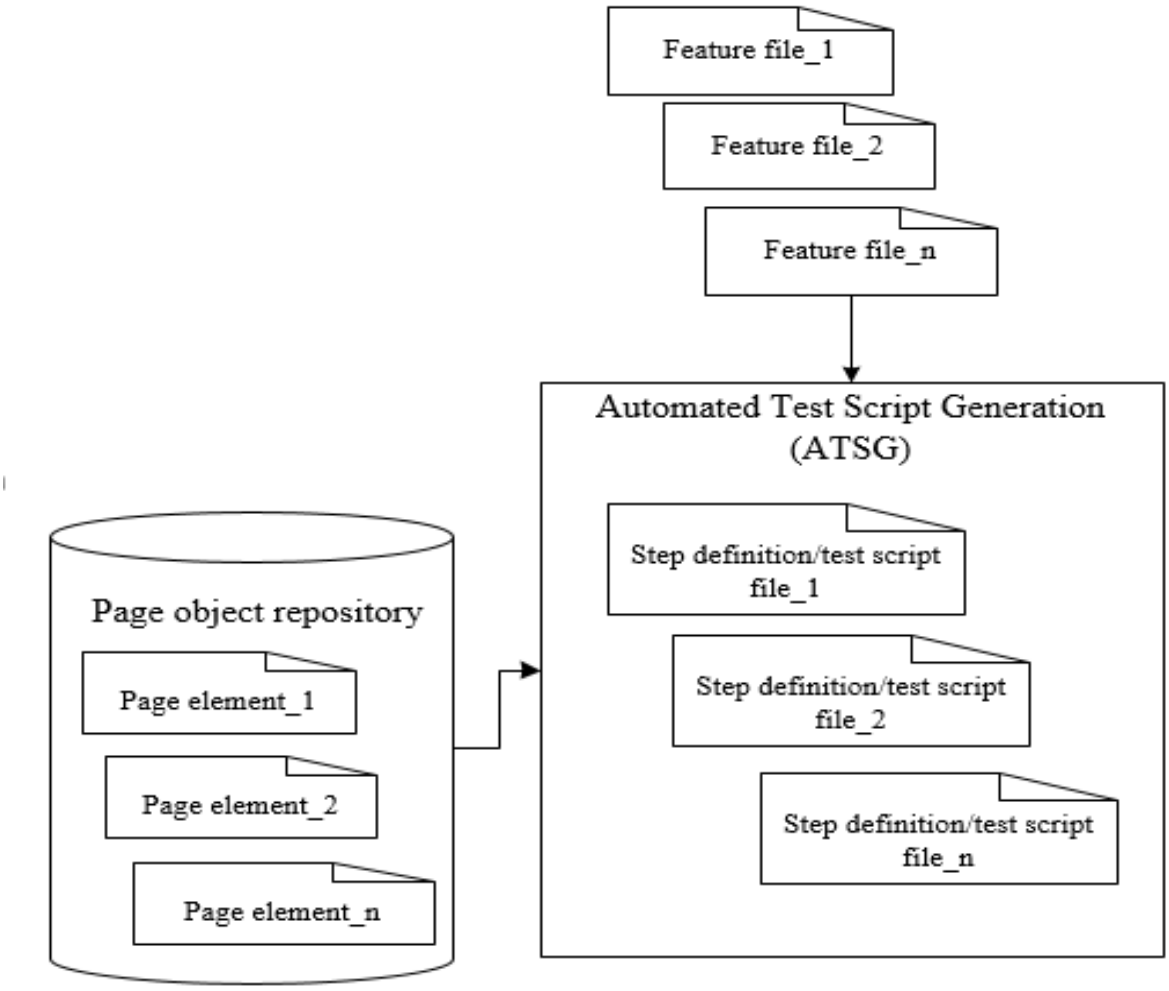

Figure 4.2 Automated test script generation component

Pages Object Repository: This component is used to store all the locators of web page elements and the methods that operate on those web elements to promote reusability and maintainability. After checking the maturity of the application, the tester deploys on a test environment and runs the application then inspects it to find all DOM elements of the web page using different locators such as the name of the element, id, CSS, XPath, link text attribute, input type, and store them in one place and used later during test design. A page object is a class that represents a page in the web application and is available to test scripts. This way of organizing the framework will simplify the task of maintaining and repairing the test scripts. For Example, the previous version of the web application contains the 'Login' button. In the next version of the web application 'Login' button changed to 'Login Now', so it is required to change all the test scripts that contain the 'Login' button. To avoid such kinds of problems, we have implemented an object repository that contains a page element locator of AUT. Whenever the tester writes the test script, the tester will

use the information to locate the web page element and needs to change only the page object repository. It is an input to test scripts /step definition generation component.

Version Controlling: This component consists of a tool that is used to control and manage automation test scripts and feature files. Version control is a system that records changes to a file or set of files over time so that the tester can recall specific versions later. Just like programmers' code, automated test scripts need to be stored in a version control system. It allows the tester to revert selected files to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. It is applicable for software automation testing to share and store the automation source code and allow the testers to collaborate as a team on the same automation source code and versioning it. Version control uses a repository (a database of all the edits to, and/or historical versions/snapshots of the project) and a working directory (a copy of all the files in the project). In our framework, we use Git and GitLab for version control and integrate it with Jenkins. To maintain healthy repositories during automation script development, each automation tester writes their script by pulling or cloning updated automation script and committing to a local repository then pushing back to a central or remote repository. Figure 4.2, shows the complete workflow as each tester started by cloning/pulling or taking source code from the central repository to their local repository then adding or editing files and committing changes as they want but until they push to the remote repository the commits or changes are stored locally which means changes are not made on the remote repository.

Continues Integration: It is a software development practice where members of a team integrate their work frequently. Each integration is verified by an automated build to detect integration errors as quickly as possible. This is also applicable for software automated testing to integrate automation test scripts and feature files into to shared repository and execute on AUT centrally.

Test Execution: This component contains a tool that enables the tester to execute automation test scripts. The proposed framework is designed to execute these test scripts either on CI tools or in the tester's local development machine or IDE. Whenever the tester generates the test script and pushes it to the version control system, test execution is triggered automatically in the CI environment by taking automated source code from the central repository without human intervention. The tester can also execute their test script

in their own IDE locally and view the test result. The framework puts automated test script source code and AUT in the same browser, which gives complete control over the AUT which means no third-party browser driver needs to launch AUT. The test runner, integrated with BDD tools and automation tools, executes the test scenarios defined in the feature files. The test runner reads the feature files, matches the steps with the corresponding step definitions, and triggers the execution of the automated tests. The test runner should execute the behaviour-driven test cases written in the Given-When-Then format. It should interpret the natural language specifications and convert them into executable code or test scripts. The test runner should map the natural language steps in the behaviour-driven test cases to the corresponding code implementation or step definitions. This mapping ensures that the specified behavior is accurately executed during the test execution.

Application Under Test: It refers to an application that is being validated by the testers. The terminology is also known as the system under test or web application in our case. Before starting test automation AUT should be a successful build. After the designing and coding phase of the development cycle, the application build is created through build tools then at that time application is under test which means the application is ready for testing. The application to be tested must be stable and ready to test otherwise, test automation will not be cost-effective. The tester should check the maturity of the application for testing before starting automation testing activities. After checking the maturity of the application, the tester deploys it on a test environment and runs the application.

Report Generation: Once the test execution process finishes, the framework generates a test report for executed test cases. This report combines a summary report for management which includes several test cases executed, pass/fail, percentage, comment, and a detailed report to developers as feedback. The report can be generated as Word, chart, JSON, PDF, HTML, screenshots, or video on demand. The test runner should generate detailed and informative test result reports. It should provide clear visibility into the pass/fail status of each test case, along with any captured logs, screenshots, or error messages exported to the required place. This helps in identifying and debugging issues quickly. The test runner should provide insights into the test coverage achieved by automation tests. It should identify areas of the software that are not adequately covered by the tests, enabling teams to enhance their tests and improve the overall quality of the software.

Chapter 5: Implementation and Evaluation

5.1 Overview

In this chapter, the researcher discussed the steps taken to implement the proposed test automation framework such as experimental setup, programming language, and tools that are used to implement a framework, test result and justify the effectiveness as well as performance of our proposed framework. This study illustrates how the different components have been used to test software based on the BDD approach.

5.2 Tools and Programming Languages

Visual Studio code editor 1.8.61: used for code editor because it is a lightweight but powerful source code editor that runs on a desktop and is available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js [68].

Java Script: For programming language, javascript has been used to implement step definition or test script. Since we select cypress as a test automation tool and run our test script within the browser.

Nodejs 20.11.0: Since our test scripts are developed using cypress and require Node.js to run those developed step definition/test scripts within the browser.

Cucumber 4.3.1: used to implement the BDD approach and formulate the behavior of the AUT in the form of a feature file using Gherkins which is a plain-text language with a simple structure and designed to be easy to learn by a non-technical person. Cucumber is the primary selection for our proposed framework implementation since it supports the BDD approach, open source, is used by all project stakeholders, reduces the communication gap, data driven testing, and integration with the cypress test automation tool and Jenkins.

Cypress 12.17.4: We decided to select cypress based on tool selection criteria suggested in [69], [70] [71] [5]. Cypress is a next-generation front-end testing tool built for the modern web and addresses the key pain points developers and QA engineers face when testing modern applications [72]. Some of the key features of cypress are: -

- Provides real-time feedback on test results and application behavior when executing the test scripts. That helps the testers identify and fix the issues.

This feature helps to improve software maintainability by ensuring that issues are fixed as soon as they are detected.

- Provides clear and detailed error messages for testers to identify the root cause of issues quickly.
- Can be integrated with the CI/CD pipelines to automate the testing process. When automating the testing process, testers can identify issues early in the development process, making it easier and less costly to fix them.
- Enables tester to design test cases/scripts for all levels of tests such as end-to-end, system, integration, and unit tests.
- Simplifies debugging tests because Cypress provides a visual interface to indicate all tests and which commands are running, passed, or failed.

Jenkins 2.401.3: used for CI, it is an open-source continuous integration/continuous delivery and deployment (CI/CD) automation software DevOps tool written in the Java programming language [73]. For test automation, it is used to execute automation tests/scripts by pulling automation source code repository from version controlling system or local workspace and promote continuous testing.

Git 2.41.0 and GitHubVersion1.37.3.1: for version controlling of our test automation source code. It is easy to integrate with CI tools, free service, and an Open-source code repository.

Step definition generator 2.2.23: used to generate step definition for each feature file written using the gherkins language. It is a visual studio code extension and easy to generate step definitions or test scripts for each feature file automatically.

AUT: We use "OrangeHRM" which is a popular open-source human resource management system to implement our proposed framework. It offers various features such as employee information management, leave and attendance tracking, performance management, and recruitment management. With its user-friendly interface and customizable modules. A fundamental feature of many web applications is the login function. For this study, we will test the login functionality of the OrangeHRM that accessed at https://opensource-demo.orangehrmlive.com/.

Cucumber test reporter 5.7.6: used to generate test reports after test execution through CI tools or local IDE.

Hardware and operating system: We used HP Windows 10, Processor Intel(R) Core (TM) i7-8565U CPU @ 1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s) to develop a framework.

5.3 Prototype Development

A prototype is developed to implement the proposed framework and justify the research. This could include developing different components of the framework, creating sample feature files, building sample test scripts, and demonstrating the potential benefits of the framework.

Configure The Framework: The framework is configured using *cypress.config.js* and *package.json* files. Figure 5.1, shows how the framework has been initially configured to conduct automation testing. For example, spec patter=cypress/e2e/HRMS/*.feature which indicates where all feature files reside and start reading from when test executions started.

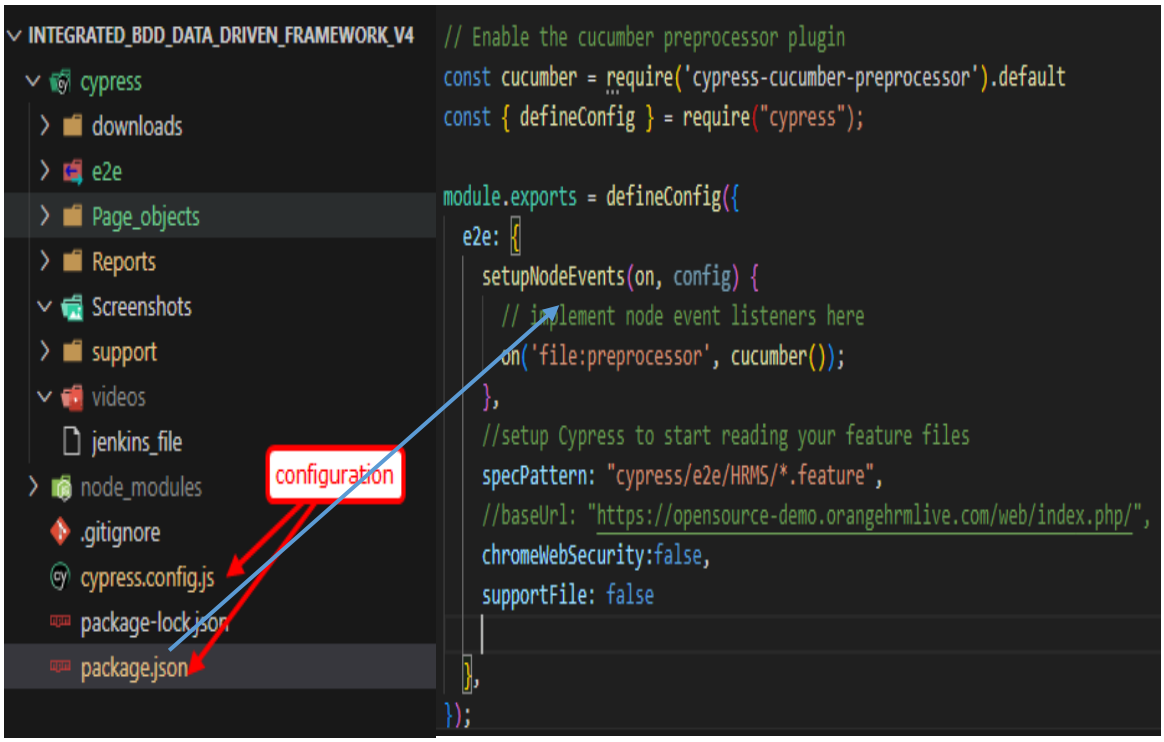

Figure 5.1 Framework configuration file

AUT Deployment: We use OrangeHRM to implement the framework which is an open-source web application so, no need to deploy it to our environment just access it online. The login page has been selected since it is the basic entry point and a repetitive task for most web applications. As indicated in Figure 5.2 the login page is ready for automation testing.

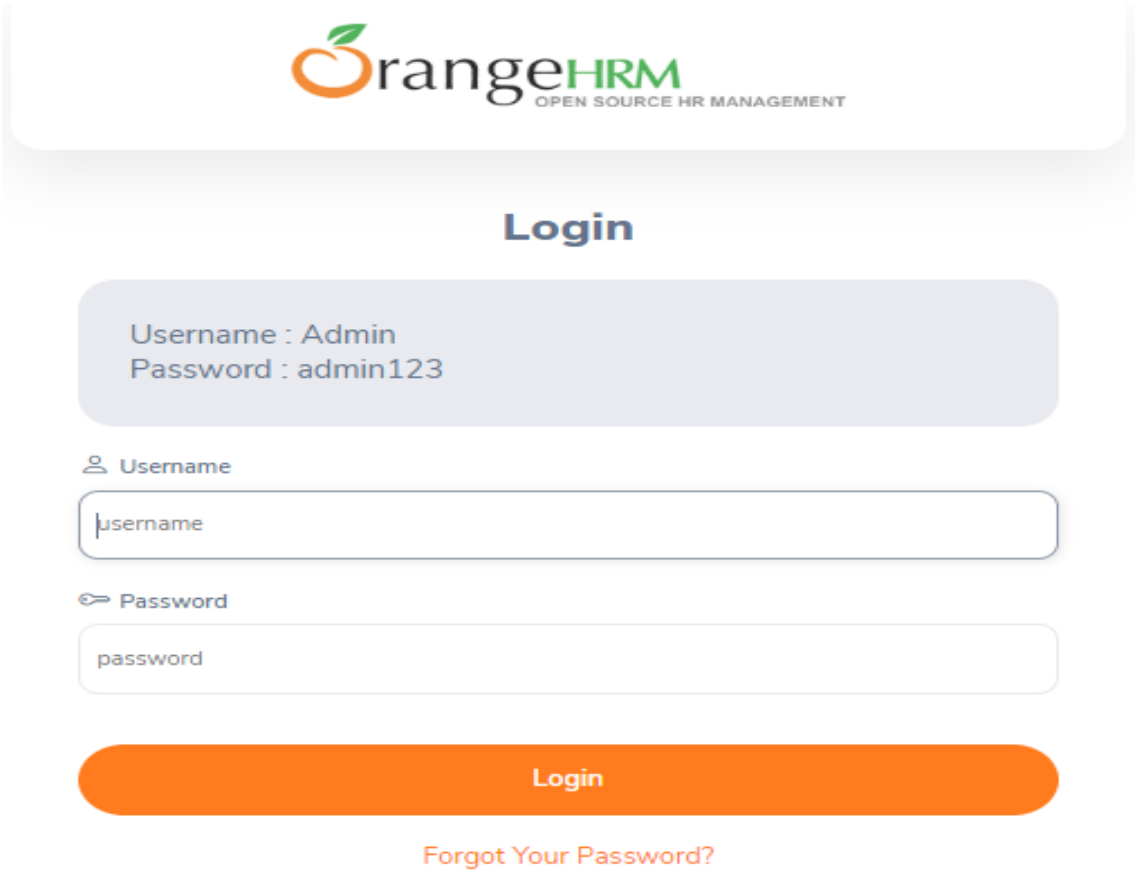

Figure 5.2: AUT ready for test automation

Feature File Creation with Test Data: As shown in Figure 5.3, a feature film for the orageHRM login page has been created using cucumber based on the feature file component template described in chapter 4 of the thesis which indicates the behavior of the login page along with test data.

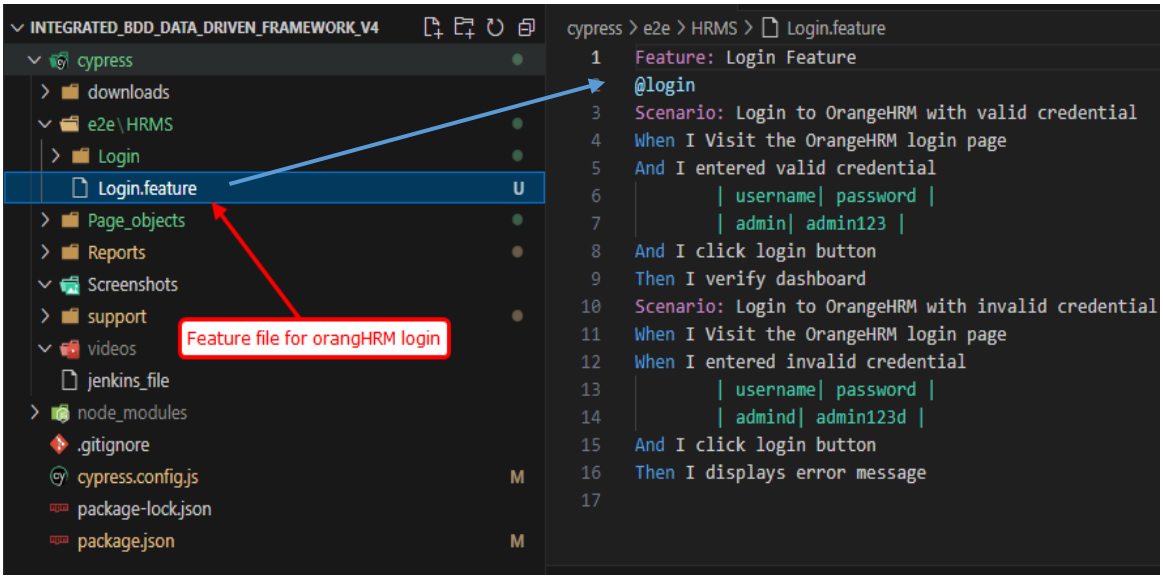

Figure 5.3 Feature file creation using cucumber

Object Repository Creation: Figure 5.4, shows how the object of the login page has been organized or created and is available for test script or step definition development.

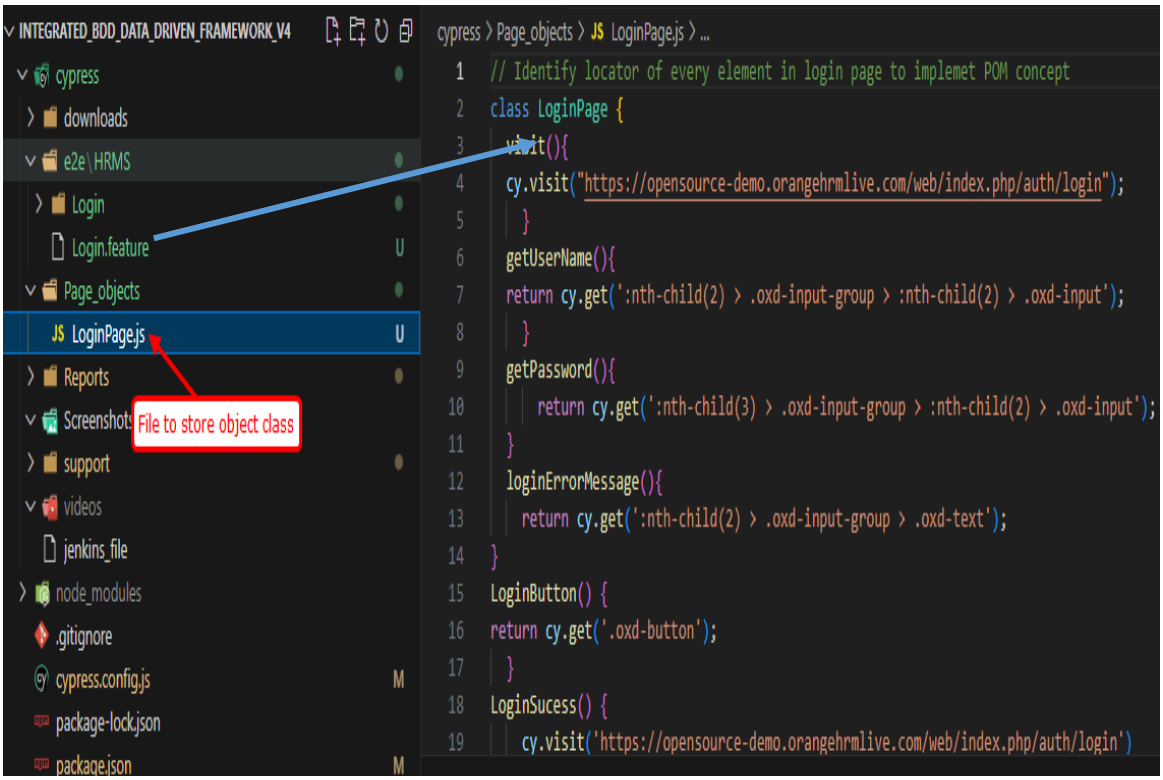

Figure 5.4: object repository for login page

Automated Test Script/Step Definition Generation: Automatic generation of test scrip from feature file using generation plugin and making some modifications. Figure 5.5, shows how step definition has been created for feature files using a step definition generator.

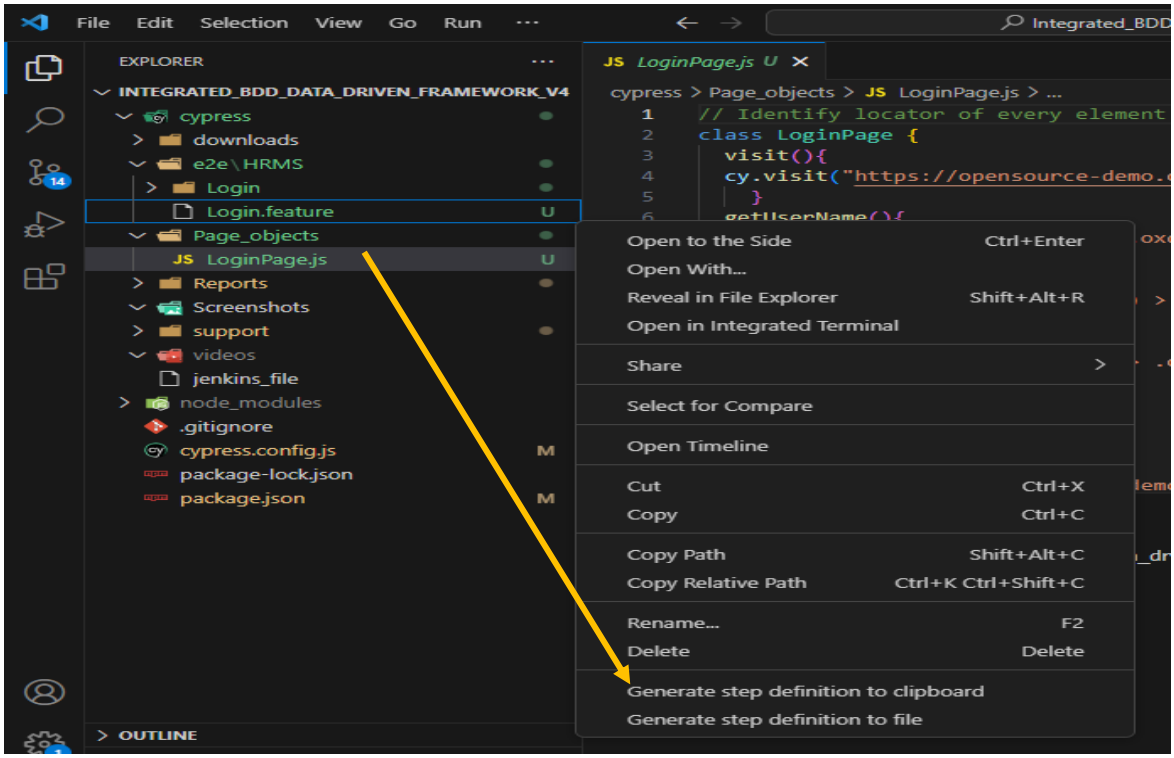

Figure 5.5 Step definition generation from feature file

As shown in Figure 5.6, the Step definition for the login feature has been created automatically based on the feature file structure.

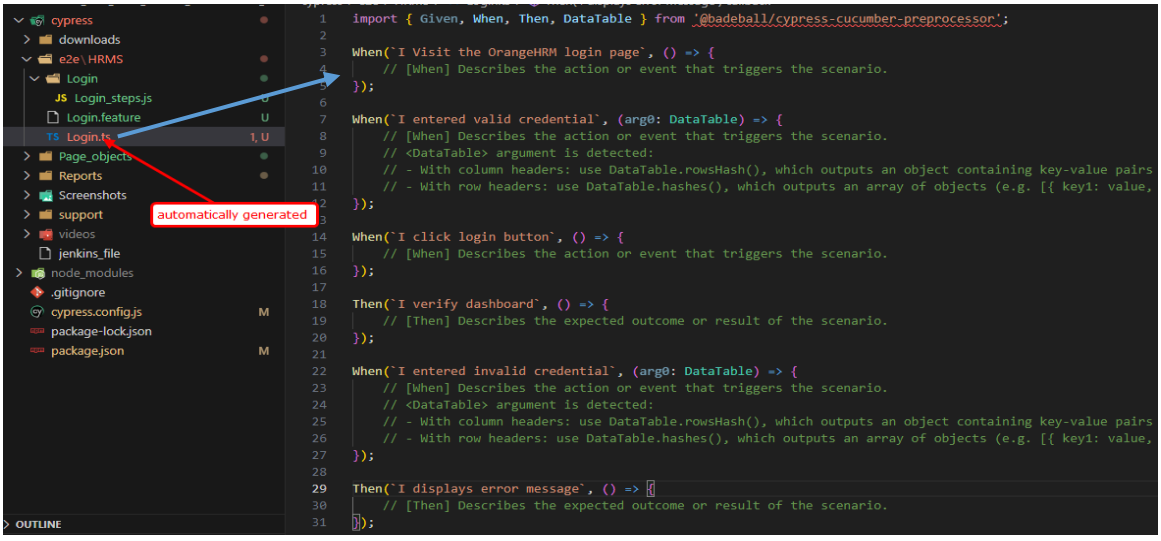

Figure 5.6: Automatically generated step definition for oragHRM login page that shows the structure of feature file

Then as shown in Figure 5.7, automated test scripts are created using cypress inside each key to map the steps in the feature files and perform action on the AUT.

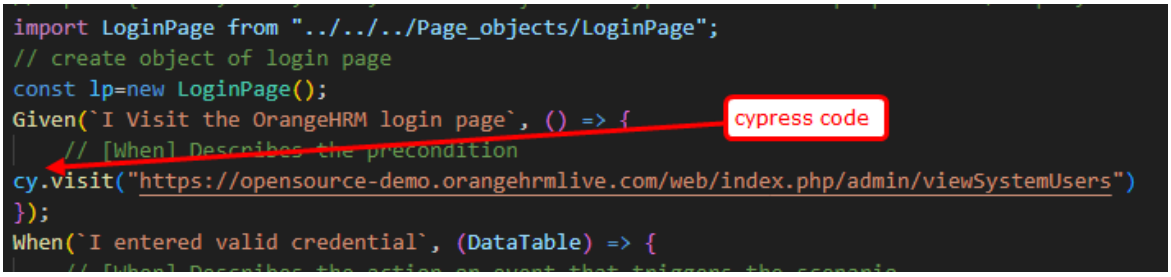

Figure 5.7 Cypress code is added for each generated feature file steps

Repository Creation for Automation Source Code Using GitHub: Install the git client on the local machine and create a remote repository using GitHub. As shown in Figure 5.8, the remote or central repository is created, and the URL that is used to push/pull/clone it is available for testers. Now each tester should have their local repository on their machines then design their test script and commit changes on automation source code to the local repository. Finally, push to a central repository to maintain complete automation source code to conduct central execution and control the version of this code centrally. This is a continuous process as long as the project has not reached final release or closed. We have to add a webhook and configure it on Git Hub which is used to integrate with Jenkins. Then GitHub webhook can send events to Jenkins whenever the push events occur in the GitHub repository.

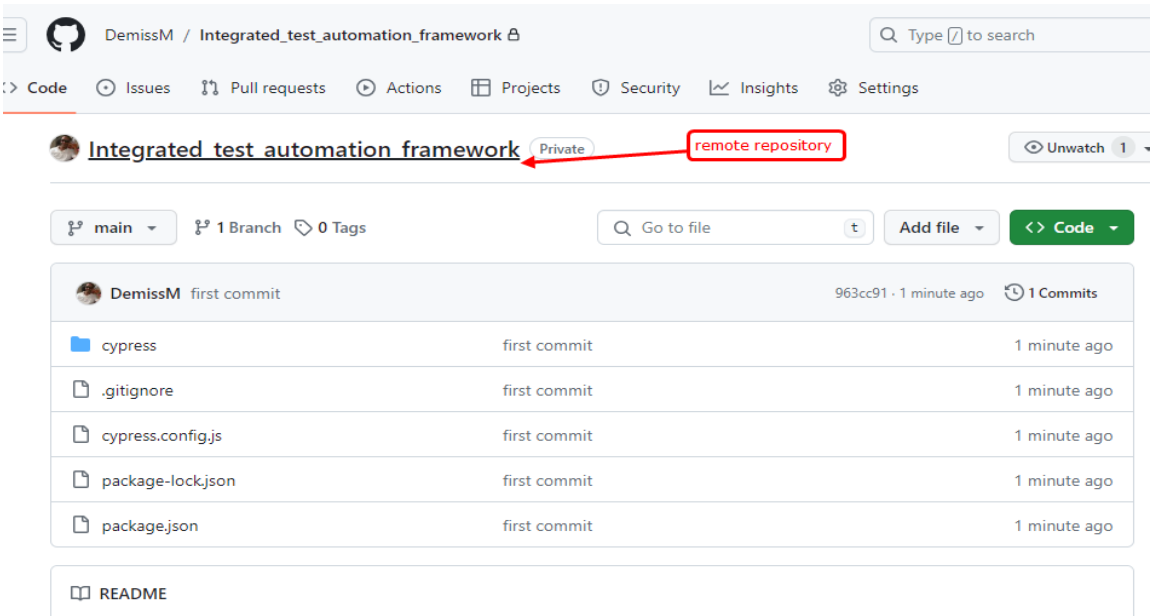

Figure 5.8 Central /remote repository for automation source code

Continuous Integration Job Creation: Install Jenkins on Windows and create a test automation project. Figure 5.9, shows the Jenkins home page with a project created and ready for further configuration.

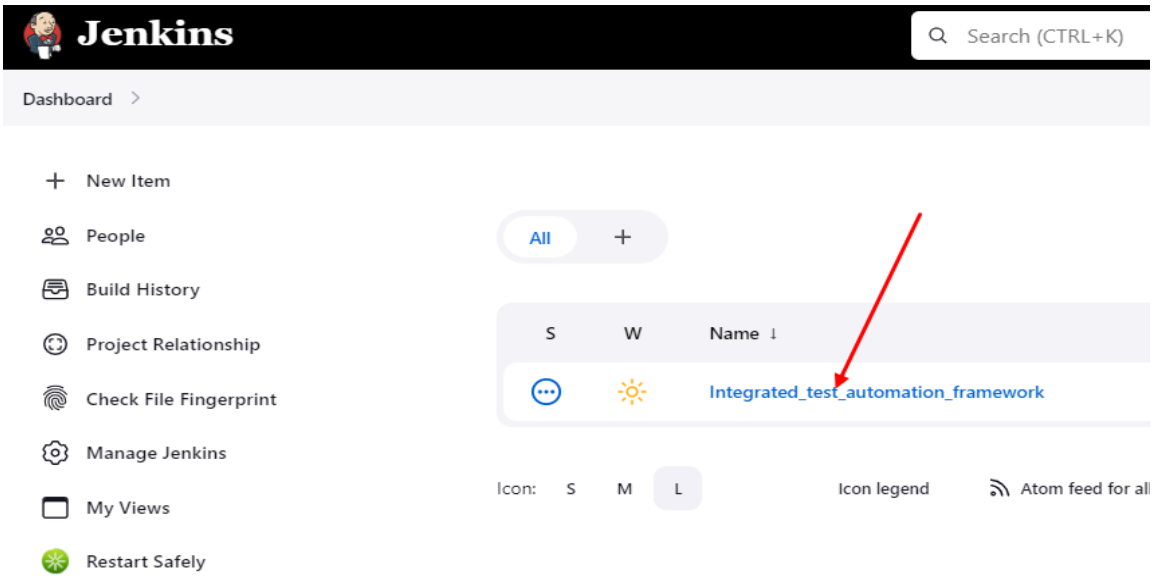

Figure 5.9 Create a new freestyle automation job/project in Jenkins

Add the automation project repository URL into to source code management section to integrate Jenkins with GitHub and get automation source code from the GitHub repository to execute the automation test script automatically. Figure 5.10, shows how the central repository URL is configured in Jenkins.

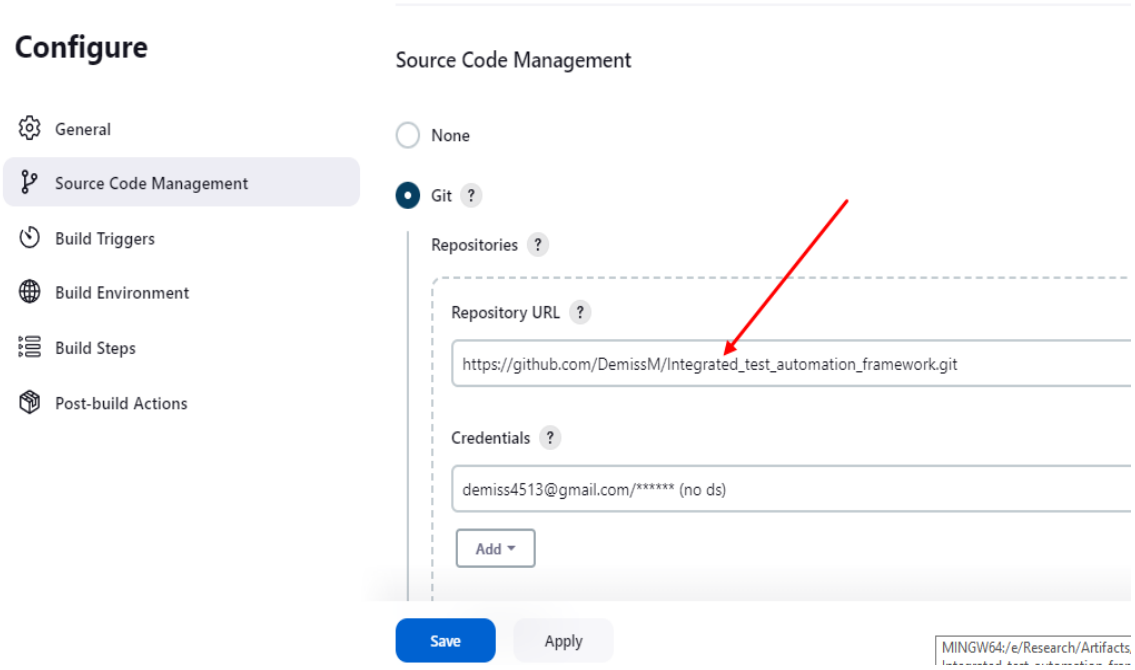

Figure 5.10 Version controlling/source code management configuration in Jenkins to get an automated source code repository.

As shown in Figure 5.11 the cypress command should be added under Jenkins's build steps to initiate cypress test execution on Jenkins.

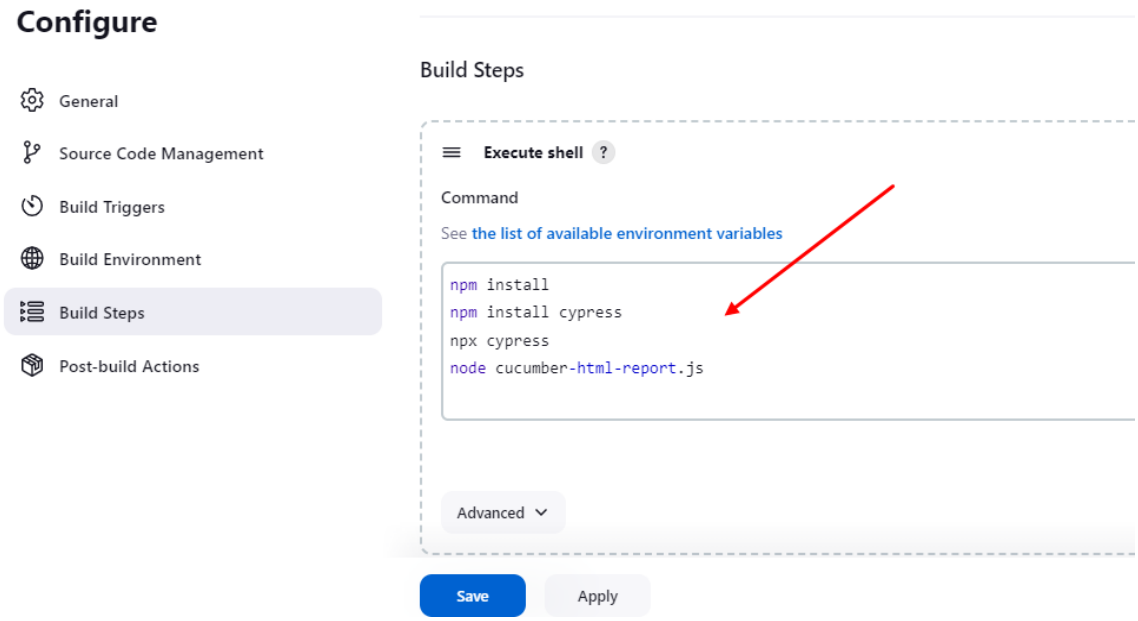

Figure 5.11 Cypress command configured in Jenkins to execute automation test scripts

Execute Automated Tests: To start test execution just navigate to the "integrated_test_automation_framework" project and click "Build Now" Then Jenkins automatically gets/clones project automation source code from Git Hub and starts execution. Execution is started whenever new automation code or change is pushed to the central/remote repository in GitHub from the testing team. During execution, the framework is responsible for reading the feature file as well as test data from the feature file folder and executing the feature file using step definition programmatically.

Report Generation: The report is automatically generated after the test execution is completed and found inside the report folder of the proposed framework. Table 5.1, shows the generated reports for the OrangeHRM login page feature test result on the Jenkins dashboard using the cucumber test reporter. It shows number and status of steps related to login feature, scenarios and duration to execute feature. As the report shows the final result of the login feature has been failed because if there is one scenario is failed with the feature the final result become failed. As indicated in the Table 5.1, the total duration to execute one feature that has total 8 steps and 2 scenarios is **6.676 ≈7**(s).

Table 5.1 Test result summary

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Login Feature	5	0	2	0	1	8	1	1	2	6.676	Failed

Figure 5.12 shows the details of individual features, scenarios and steps along with duration. As indicated in the Table 5.1, the total duration **6.676 ≈7**(s) is calculated from individual scenarios and related steps shown in Figure 5.12.

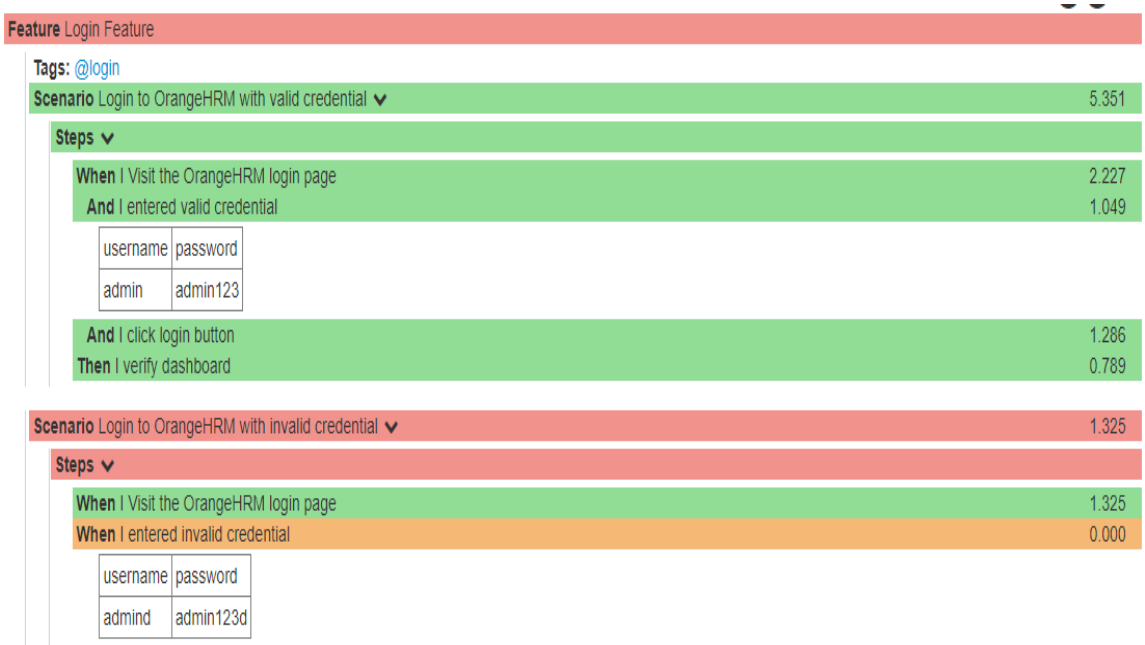

Figure 5.12 Detail report for login feature

5.4 Test Result Analysis

Test execution time: This metric measures the time it takes to execute automated tests. Faster test execution times allow for more frequent testing and faster feedback. As the test execution result indicates in Table 5.1 execution took approximately 7 seconds to execute single test case for each functional, regression, and acceptance testing. This shows our framework was well designed and implemented in terms of tool selection and structure of the framework. Enable the tester to execute the test within a short period and faster feedback to the developer.

As indicated in Figure 5.13 our framework is faster than the existing one to execute a single test case. The time taken to execute functional, regression, and acceptance testing is the same because there is no change on the test case which means once the test case is developed, we can execute multiple times as we want for different types of testing.

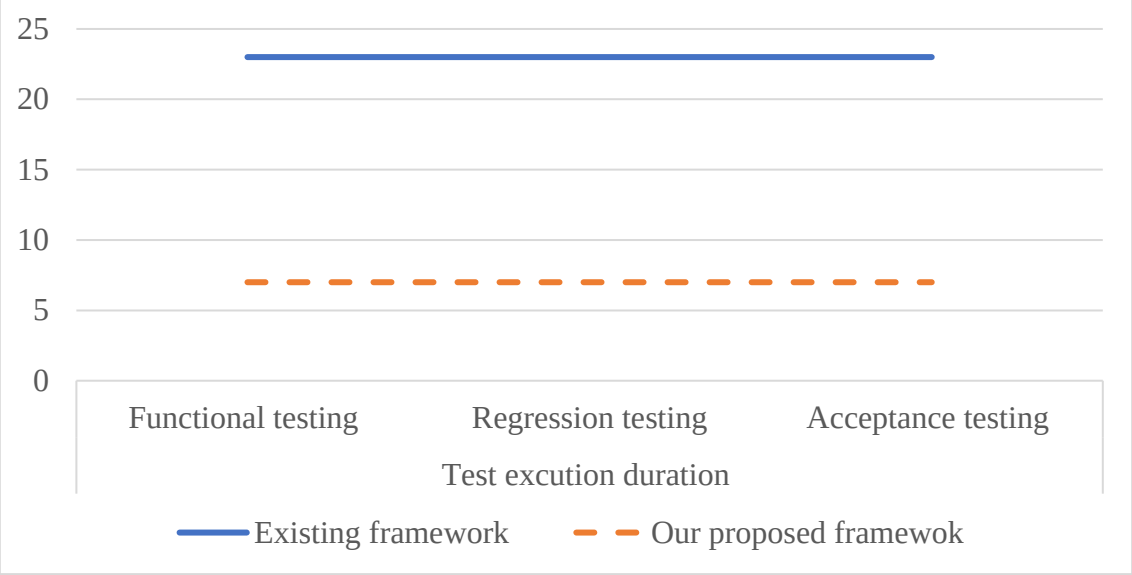

Figure 5.13 Comparison of our framework against existing in terms of test execution time

5.5 Expert Evaluation

Most of the research that produces frameworks as artifacts uses expert evaluation to judge effectiveness of the artifact. Accordingly, we conduct this method to evaluate the proposed framework's effectiveness. Practical demonstrations and presentations are conducted to domain experts using Zoom virtual meetings to evaluate the framework. A virtual meeting is preferred due to time and resource constraints to conduct the demonstration by arranging a test environment that showcases the features and capabilities of the framework. Following the presentation, the survey questionnaires were prepared as indicated in annex A based on the objective of the research and distributed to 12 randomly selected participants among testers to gather their attitude and feedback on the proposed framework regarding the suitability for BDD, functionalities, integration and content of the framework. All the invited 12 participants completed the survey. Hence, the response rate of the framework evaluation survey is 100%. The survey data is registered and analysed using SPSS software. The questionnaires are organized as a 5-point Likert-type scale and the attitude or response of the participant is quantified as strongly disagree=1, disagree=2, neutral=3, agree=4, and strongly agree=5 Based on the given scale framework evaluators provide a value for each question. The questionnaires are depicted in Annexes A.

Table 5.2 Expert evaluation result

Evaluation Criteria	**User Response in Frequency**				
	Strongly disagree	Disagree	neutral	agree	Strongly agree
Suitability					
The framework supports the key principles and practices of BDD such as focusing on features that deliver business value, working together to specify behaviors of the system (collaboration), mitigating uncertainty, shared understanding of everything related to the software to be developed, and delivering living documentation. providing faster feedback to developers	-	-	-	10	2
% Suitability	-	-	-	**83**	17
Integration					
The framework effectively integrated with IDE, version control systems, continuous integration, BDD, automation testing, and test reporting tools.	-	-	1	9	2
% Integration	-	-	8	**75**	17
Functionalities					
The framework can translate human-readable scenarios or behavior of the software into executable test code through automatically generated step definitions, used to perform requirement analysis, retesting, regression testing, acceptance testing, automatic test execution, and reporting. Aligning automated tests with business requirements and user expectations.	-	-	-	11	1
% Functionality	-	-	-	**92**	8
Content of the framework					
The content of the framework is reusable, maintainable, and usable enough, and cover all relevant components of a typical framework.	-	1	-	9	2
Content of the framework %	-	8	-	75	17
Total	-	1	1	39	7
Average	-	1	1	10	2
% Cumulative	-	8	8	**83**	17

As shown in Table 5.2 the survey result is analysed as follows:

- 83% and 17% of the respondents agree and strongly agree respectively on the suitability of the framework for the BDD approach which means the proposed smoothly integrates with BDD and enables the tester to effectively implement BDD practices, streamline the testing process, improve collaboration, and deliver high-quality software that meets user expectations. There is no respondent with the rest of the scale.
- 75% and 17% of the respondents agree and strongly agree on the integration of BDD, IDE, test automation testing, version controlling, continuous integration, and reporting tools. There is one respondent who remains neutral.
- 92% and 8% of the respondents agree on the basic functionality of the framework and there is no respondent with the rest of the scales
- 75% and 17 % of the respondents agree on the content of the proposed framework. This confirms us our framework is easy to use, maintainable, usable, reusable, and contains the required components for test automation.

As indicated in the cumulative result **83 %** of the respondents have agreed on the suitability, integration, functionality, and content of the proposed framework but the framework needs some improvement to enhance it capability to conduct automation testing.

Chapter 6: Conclusions, Contribution, and Future Work

6.1 Conclusions

In this research work, we have conducted a literature review on software testing, known software development methodologies, and a detailed discussion on the general principle, practice, and process of the BDD approach, automation testing, and framework for automation testing. we survey the state-of-the-art automation software testing framework to identify their gap and review related work on different automation framework techniques that are employed for software automation testing activities.

The general objective of this research to design and develop an integrated software testing automation framework that supports the BDD software development approach. Our framework was designed and implemented to address the previous framework gap by integrating BDD, IDE, version controlling, continuous integration, test automation and reporting components. The proposed framework was implemented using cucumber, cypress, visual studio code, Nodejs, Jenkins, JavaScript, Github, step-definition generator, and cucumber test reporter. Our proposed framework was evaluated using experiment and user acceptance testing. The test result shows improvement in test execution time as compared to the existing framework. Expert evaluation result shows that 83% of the respondents agreed on the suitability, integration, functionality, and content of the framework. The proposed framework offers a valuable solution for organizations seeking to adopt BDD methodologies and improve their software test automation processes. In the future, it is important to add artificial intelligence capability to improve the framework.

6.2 Contribution of the study.

- ✓ Effective integration of tool required for automated software testing task.
- ✓ Improve test execution speed for regression and retesting.
- ✓ Promote reusability of the framework component which means once a framework is designed, we can use it to automated testing of any application. Just a few changes in configuration, test data & object repository are required.

6.3 Future Work

The future works we recommend to extend the capability of the proposed framework are:

- ✓ The framework was designed to test web applications only so, we suggest other researchers evaluate our framework and incorporate a feature that needs to test mobile and other application types.
- ✓ Incorporate artificial intelligence feature

References

[1] D. DeVolder, S. Ghazanshahi, and J. Zadeh, "Software testing and quality assurance," in *WMSCI 2008 - The 12th World Multi-Conference on Systemics, Cybernetics and Informatics, Jointly with the 14th International Conference on Information Systems Analysis and Synthesis, ISAS 2008 - Proc.*, EXCEL BOOKS PRIVATE LIMITED, 2008, pp. 105–108. doi: 10.1002/9780470382844.

[2] S. Engineering Standards Committee of the IEEE Computer Society, *IEEE Std 829^{TM}-2008 IEEE Standard for Software and System Test Documentation IEEE Computer Society*, vol. 2008, no. July. 2008.

[3] D. Galin, *Software quality assurance: from theory to implementation*. Pearson Education Limited, 2004.

[4] SWEBOK, *Guide to the Software Engineering Body of Knowledge.* IEEE Computer Society, 2004.

[5] Mekuanet Tilahun and Mekuanint Agegnehu "Designing and Developing An Integrated Automation Software Testing Framework, unpublished Masters Thesis," MSc, Bahir Dar University, 2020.

[6] 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.

[7] "Behavior-driven development – cucumber." Accessed: Aug. 31, 2023. [Online]. Available: https://cucumber.io/docs/bdd/.

[8] J. Ferguson and D. North, "BDD_in_Action." , Shelter Island , NY 11964,Manning publication.

[9] "Test Automation Frameworks." Accessed: Jun. 01, 2021. [Online]. Available: https://smartbear.com/learn/automated-testing/test-automation-frameworks/

[10] R. S. August, "Building a Customized Test Automation Framework Using Open Source Tools," 2016, *Rapidvalue solution, Pleasanton (USA)*.

[11] R. Anand and M. Arulprakash, "Business driven automation testing framework," *International Journal of Engineering & Technology*, vol. 7, no. 2.8, p. 345, Mar. 2018, doi: 10.14419/ijet.v7i2.8.10438.

[12] M. Irshad, R. Britto, and K. Petersen, “Adapting Behavior Driven Development (BDD) for large-scale software systems,” *Journal of Systems and Software*, vol. 177, Jul. 2021, doi: 10.1016/j.jss.2021.110944.

[13] A. Sheshasaayee and P. Banumathi, “Impacts of behavioral driven development in the improvement of quality software deliverables,” *Proceedings of the 3rd International Conference on Inventive Computation Technologies, ICICT 2018*, pp. 228–230, 2018, doi: 10.1109/ICICT43934.2018.9034287.

[14] M. A. Umar and C. Zhanfang, “A Study of Automated Software Testing : Automation Tools and Frameworks,” *International Journal of Computer Science Engineering (IJCSE)*, vol. 8, no. 06, pp. 217–225, 2019, doi: 10.5281/zenodo.3924795.

[15] Z. Sun, Y. Zhang, and Y. Yan, “A Web Testing Platform Based on Hybrid Automated Testing Framework,” *Proceedings of 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference, IAEAC 2019*, no. Iaeac, pp. 689–692, 2019, doi: 10.1109/IAEAC47372.2019.8997684.

[16] M. Hanna, A. E. Aboutabl, and M. M. Mostafa, “Automated Software Testing Framework for Web Applications,” *International Journal of Applied Engineering Research ISSN*, vol. 13, no. 11, pp. 9758–9767, 2018.

[17] Shimelis Tamiru and Getahun Semeon “An Integrated Software Test Process Framework: The Case of Selected Ethiopian Software Companies,” MSc, St.Mary’s University, 2017.

[18] S. Coordinating and C. Society, “INTERNATIONAL STANDARD ISO / IEC / IEEE Software and systems engineering —Test,” 2013. [Online]. Available: ieee.com

[19] E. V Veenendaal, “Standard glossary of terms used in Software Testing, Version 1.2,” 2010.

[20] J. Kasurinen, “Software Organizations and Test Process Development,” no. Myers 2004, 2007.

[21] S. Coordinating and C. Society, “INTERNATIONAL STANDARD ISO / IEC / IEEE Software and systems engineering —Part 3: Test documentation,” 2013. [Online]. Available: ieee.com

[22] S. Coordinating and C. Society, “INTERNATIONAL STANDARD ISO / IEC / IEEE Software and systems engineering —Part 2: Test process,” 2013.

[23] International Software Testing Qualifications Board, “Certified Tester Foundation Level Syllabus,” 2018.

[24] “Static Testing vs Dynamic Testing: What’s the Difference?” Accessed: May 07, 2022. [Online]. Available: https://www.guru99.com/static-dynamic-testing.html

[25] “What is Functional Testing? Types & Examples (Complete Tutorial).” Accessed: Mar. 20, 2022. [Online]. Available: https://www.guru99.com/functional-testing.html

[26] D. DeVolder, S. Ghazanshahi, and J. Zadeh, *Software testing and quality assurance*, vol. 1. A JOHN WILEY & SONS, INC., PUBLICATION, 2008. doi: 10.1002/9780470382844.

[27] “What is Integration Testing? - Definition from Techopedia.” Accessed: May 13, 2021. [Online]. Available: https://www.techopedia.com/definition/7751/integration-testing

[28] “Integration Testing - Big Bang, Top Down, Bottom Up & Hybrid Integration - Software Testing Material.” Accessed: May 13, 2021. [Online]. Available: https://www.softwaretestingmaterial.com/integration-testing/

[29] “Big-Bang Testing - Tutorialspoint.” Accessed: May 18, 2021. [Online]. Available: https://www.tutorialspoint.com/software_testing_dictionary/big_bang_testing.htm

[30] “Bottom Up Testing - Tutorialspoint.” Accessed: May 18, 2021. [Online]. Available:https://www.tutorialspoint.com/software_testing_dictionary/bottom_up_testing.htm

[31] “Integration Testing - Big Bang, Top Down, Bottom Up & Hybrid Integration - Software Testing Material.” Accessed: May 28, 2021. [Online]. Available: https://www.softwaretestingmaterial.com/integration-testing/

[32] M. Atifi, A. Mamouni, and A. Marzak, “A comparative study of software testing techniques,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10299 LNCS, no. 6, pp. 373–390, 2017, doi: 10.1007/978-3-319-59647-1_27.

[33] A. Verma, A. Khatana, and S. Chaudhary, “A Comparative Study of Black Box Testing and White Box Testing,” *International Journal of Computer Sciences and Engineering*, vol. 5, no. 12, pp. 301–304, 2017, doi: 10.26438/ijcse/v5i12.301304.

[34] B. Homès, *Fundamentals of Software Testing*.

[35] “What is Waterfall Model in software testing and what are advantages and disadvantages of Waterfall Model.” Accessed: Dec. 22, 2021. [Online]. Available: https://testingfreak.com/waterfall-model-software-testing-advantages-disadvantages-waterfall-model/

[36] “V-Model in Software Testing.” Accessed: May 30, 2021. [Online]. Available: https://www.guru99.com/v-model-software-testing.html

[37] “SDLC - V-Model.” Accessed: Dec. 22, 2021. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[38] D. Hamlet, “Foundations of software testing,” *ACM SIGSOFT Software Engineering Notes*, vol. 19, no. 5, pp. 128–139, 1994, doi: 10.1145/195274.195400.

[39] “(No Title).” Accessed: May 30, 2021. [Online]. Available: https://www.tutorialspoint.com/sdlc/pdf/sdlc_iterative_model.pdf

[40] “SDLC-ITERATIVE MODEL.” Accessed: May 30, 2021. [Online]. Available: http://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

[41] R. Khan, A. K. Srivastava, and D. Pandey, “Agile approach for Software Testing process,” *Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016*, no. January, pp. 3–6, 2017, doi: 10.1109/SYSMART.2016.7894479.

[42] “Agile Testing - Methodologies.” Accessed: Dec. 23, 2021. [Online]. Available: https://www.tutorialspoint.com/agile_testing/agile_testing_methodologies.htm

[43] “BDD (Behavior Driven Development) Framework: A Complete Tutorial.” Accessed: Aug. 31, 2023. [Online]. Available: https://www.softwaretestinghelp.com/bdd-framework/

[44] A. Musliu and X. Jashari, “Software Automated Testing using BDD Approach with Cucumber Software Automated Testing using BDD Approach with Cucumber Framework Framework Software Automated Testing using BDD Approach with

Cucumber Framework.” [Online]. Available: https://knowledgecenter.ubt-uni.net/conferencehttps://knowledgecenter.ubt-uni.net/conference/2021UBTIC/all-events/409

[45] “Living document” Accessed: Dec. 02, 2023. [Online]. Available: https://support.smartbear.com/cucumberstudio/docs/bdd/living-doc.html.

[46] “Behaviour-Driven Development - Cucumber Documentation.” Accessed: Sep. 02, 2023. [Online]. Available: https://cucumber.io/docs/bdd/

[47] “Behaviour Driven Development in Agile — a Step-by-Step Guide to start | by Anca Onuta | Geek Culture | Medium.” Accessed: Sep. 02, 2023. [Online]. Available: https://medium.com/geekculture/behaviour-driven-development-in-agile-a-step-by-step-guide-to-start-8ffac4763a7

[48] “The Definitive Guide to Behavior Driven Development (BDD).” Accessed: Sep. 02, 2023. [Online]. Available: https://www.spiceworks.com/tech/devops/articles/what-is-bdd/

[49] “BDD Helps You Find Bugs Before They Find You. Start learning BDD!” Accessed: Dec. 05, 2023. [Online]. Available: https://specflow.org/learn/bdd/#3-phases

[50] “User Stories | Examples and Template | Atlassian.” Accessed: Dec. 03, 2023. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories

[51] “BDD for Beginners 2 | How BDD starts in an Organisation | Sample BDD Workshop - YouTube.” Accessed: Dec. 03, 2023. [Online]. Available: https://www.youtube.com/watch?v=eRSr2lpHpwI

[52] “User Story Template for Agile | Agile Alliance.” Accessed: Dec. 05, 2023. [Online]. Available: https://www.agilealliance.org/glossary/user-story-template/

[53] “User Stories | Examples and Template | Atlassian.” Accessed: Dec. 05, 2023. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories

[54] “Writing scenarios with Gherkin syntax | CucumberStudio Documentation.” Accessed: Sep. 02, 2023. [Online]. Available:

https://support.smartbear.com/cucumberstudio/docs/bdd/write-gherkin-scenarios.html

[55] "Behavior Driven Development - Introduction." Accessed: Dec. 03, 2023. [Online]. Available: https://www.tutorialspoint.com/behavior_driven_development/behavior_driven_development_introduction.htm

[56] "Automated Testing Process - Tutorialspoint." Accessed: Jun. 01, 2021. [Online]. Available: https://www.tutorialspoint.com/qtp/qtp_test_automation_process.htm

[57] "What is framework? - Definition from WhatIs.com." Accessed: May 30, 2021. [Online]. Available: https://whatis.techtarget.com/definition/framework

[58] "A Guide to Automation Frameworks |Smartsheet." Accessed: Jun. 01, 2021. [Online]. Available: https://www.smartsheet.com/test-automation-frameworks-software

[59] M. A. Umar and C. Zhanfang, "A Study of Automated Software Testing: Automation Tools and Frameworks," *International Journal of Computer Science Engineering (IJCSE)*, vol. 8, no. 06, pp. 217–225, 2019, doi: 10.5281/zenodo.3924795.

[60] C. Chandraprabha, A. Kumar, and S. Saxena, "Data Driven Testing Framework using Selenium WebDriver," *Int J Comput Appl*, vol. 118, no. 18, pp. 18–23, doi: 10.5120/20845-3497.

[61] S. Lamba, V. Rishiwal, and A. Rana, "AN AUTOMATED DATA DRIVEN CONTINUOS TESTING FRAMEWORK," 2015.

[62] K. V Arya and H. Verma, "Keyword Driven Automated Testing Framework for Web Application.".

[63] "Keyword Driven Framework Example | Selenium Easy." Accessed: Nov. 25, 2023. [Online]. Available: https://www.seleniumeasy.com/selenium-tutorials/keyword-driven-framework-example

[64] S. Tyagi, R. Sibal, B. Suri, B. Wadhwa, and S. Shekhar, "Development of reusable hybrid test automation framework for web-based scrum projects," *Journal of Applied Science and Engineering*, vol. 21, no. 3, pp. 455–462, doi: 10.6180/jase.201809_21(3).0017.

[65] X. Xie, Z. Yang, J. Yu, and W. Zhang, “Design and implementation of bank financial business automation testing framework based on QTP,” in *Proceedings of 2016 5th International Conference on Computer Science and Network Technology, ICCSNT*, Changchun, China: IEEE, 2017, pp. 143–147. doi: 10.1109/ICCSNT.2016.8070136.

[66] M. N. Islam and S. M. K. Quadri, “Framework for automation of cloud-application testing using selenium (FACTS),” *Advances in Science, Technology and Engineering Systems*, vol. 5, no. 1, pp. 226–232, doi: 10.25046/aj050129.

[67] S. Gojare, R. Joshi, and D. Gaigaware, “Analysis and design of selenium webdriver automation testing framework,” in *Procedia Computer Science,* Elsevier B.V, pp. 341–346. doi: 10.1016/j.procs.2015.04.038.

[68] “Documentation for Visual Studio Code.” Accessed: Jun. 13, 2021. [Online]. Available: https://code.visualstudio.com/docs

[69] “Test Automation Tool Selection Criteria and Comparison Matrix (A Complete Guide).” Accessed: Dec. 17, 2023. [Online]. Available: https://www.softwaretestinghelp.com/automation-testing-tutorial-4/

[70] “How to Select Best Automation Testing Tool?” Accessed: Dec. 17, 2023. [Online]. Available: https://www.guru99.com/testing-automation-why-right-tools-are-necessary-for-testing-success.html

[71] S. Kumar, “APPLYING AUTOMATION TESTING FRAMEWORK AS AN degree of M . Sc . in Information Systems with Computing at Dublin Business School Shravan Kumar Student Id : 10319743,” 2016.

[72] “Why Cypress? | Cypress Documentation.” Accessed: Jan. 14, 2022. [Online]. Available: https://docs.cypress.io/guides/overview/why-cypress#In-a-nutshell

[73] “What is Jenkins and How Does It Work?” Accessed: May 07, 2022. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Jenkins

Annex A: Expert Evaluation Survey Questionnaire

Dear Sir or Madam:

Thank you for taking part in this survey and for contributing to my research!

Aim: This survey aims to gather your opinion about the utility and some other quality characteristics of the proposed framework.

Anonymity: The information obtained from this questionnaire will be employed just for research purposes and the anonymity of the participants is ensured. No individual information will be disclosed.

Duration: The questionnaire will take about 15 minutes to complete.

This research is believed to produce results that can improve software testing activities.

Thank you for your dedication to providing your genuine feedback regarding the proposed framework.

Thank you again!!!

Researcher:

Demiss Mammo Tadesse

Software Quality Assurance professional and student

Put (✓) for your evaluation in the corresponding box of evaluation criteria according to the following: (1) strongly disagree, (2) disagree, (3) neutral, (4) agree, and (5) strongly agree.

Table 6.1: Framework evaluation criteria

Evaluation Criteria	1	2	3	4	5
Suitability					
The framework supports the key principles and practices of BDD such as focusing on features that deliver business value, working together to specify the behavior of the system (collaboration), mitigating uncertainty, shared understanding of everything related to the software to be developed, and delivering living documentation. providing faster feedback to developers.					
Integration					
The framework effectively integrated with IDE, version control systems, continuous integration, BDD, automation testing, and test reporting tools.					
Functionalities					
The framework can translate human-readable scenarios or behavior of the software into executable test code through automatically generated step definitions, requirement analysis, retesting, regression testing, acceptance testing, automatic test execution, and reporting. Aligning automated tests with business requirements and user expectations.					
Content of the framework					
The content of the framework is reusable, maintainable, and usable enough, and cover all relevant components of a typical framework.					

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university and that all sources of materials used for the thesis have been fully acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed By Advisor:

Name: _____

Signature: _____

Date: _____