International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Enhancing Test Automation with an Advanced Page Object Model for Scalable and Maintainable Web Applications

Mohnish Neelapu

Category / Domain: QA Automation

Abstract

This research suggests an Advanced Page Object Model (POM) Design to overcome limitations of existing POM models in test automation with an aim to improve scalability, maintainability, and flexibility in complex web applications. The suggested framework provides substantial improvements like multi-layered design, run-time object creation using factory patterns, fluent interfaces to enable method chaining efficiently, component-based modular design for reusability, and data-driven testing using external configurations. All these enhancements combined make it simple for code organization, redundancy reduction, and easy flexibility to UI changes. Comparison of the Advanced POM with traditional POM frameworks shows improvements in code maintainability, reusability, execution time, and scalability, making the Advanced POM efficient and flexible to meet existing test automation needs. Benchmark results indicate code change reduction, maintenance time, run time, and enhanced test case reusability and framework scalability. Enhanced POM is especially suitable while dealing with dynamic UI elements and frequent changes, and it is an ideal solution for modern web applications.

Keywords: Automation Framework, Modular Test Automation, Page Object Model, Software Development Life Cycle and Code Reusability

I. INTRODUCTION

In the ever-evolving times of software development, test automation frameworks are an absolute component of software quality, efficiency, and reliability [1-3]. Such models allow test cases to execute automatically, with less human intervention, speeding up testing cycles, and improving defect detection in Agile and DevOps-based settings [4]. Of the various automation design patterns, POM is most popular so due to its ability to enhance maintainability and reusability of test scripts [5-6]. POM employs an automation framework by encapsulating UI actions and components associated with them into specific classes, thereby achieving a clean separation of test logic and application interactions. This modularity significantly enhances code organization, readability, and reusability [7-8]. However, in the growing web applications developed using the new-generation front-end development tools like React, Angular, and Vue.js, POM patterns based on conventional design come with enormous impediments like the ones associated with scalability, redundancy of code pattern repetition, added complexity in maintenance, and impediments related to dynamically-loaded components [9-10].

The larger the size of an automation suite, the more difficult it is to deal with a lot of page objects, resulting in excessive maintenance overhead and decreased test efficiency [11-13]. Such limitations call for an evolved strategy of POM incorporating contemporary software development practices like layering, factory patterns, component-based POM, and fluent interfaces to strengthen modularity, maintainability, and scalability [14-16]. This research plans to investigate and suggest an optimal Advanced Page Object Model Design, overcoming the drawback of traditional POM frameworks with dynamic element manipulation, reusable test components, and organized test running methodologies [17-18]. Through the application of sophisticated architectural patterns and best practices in automation, the approach here aims to make test framework performance more efficient, minimize redundancy, and better support adaptability to UI variations, making test automation more robust, scalable, and maintainable in contemporary software development environments [19-20].

A.Objectives

- Build an Advanced POM that enhances the reusability of code and reduces redundancy while automating testing.
- Decrease execution time as well as maintenance compared to usual POM patterns.
- Apply tactics like dynamic creation of objects, component-based module design, as well as layers of architecture for dealing with frequent UI changes optimally.
- Leverage fluent interfaces and factory design patterns to automate method chaining as well as script readability.
- Make a scalable testing framework that accepts external configurations along with continuous test environments.
- Compare Advanced POM with conventional POM on maintainability, scalability, execution time, and code complexity.

B.Research questions

- How does the Advanced POM enhance scalability and maintainability in comparison to conventional POM?
- What is the effect of Advanced POM on execution time and efficiency in automated test cases?
- What advantages does the use of fluent interfaces and factory patterns provide in terms of test script readability and maintenance?
- How does the Advanced POM ensure easy automation in CI/CD pipelines?
- What are the benchmarking findings between the Advanced POM versus conventional frameworks in terms of maintainability, execution time, scalability, and flexibility?

II. LITERATURE REVIEW

Ricca, F et al. [1] presents an enhanced method to the POM in their study of making web and mobile application test frameworks more scalable and maintainable. The process focuses on modularization, abstraction, parallel testing, and use of cloud tools to develop a scalable test automation framework that is able to do cross-browser as well as cross-platform testing. The major strength of this strategy is that it is able to enhance test efficiency through code minimization, reduction of redundancy, and parallel testing execution, resulting in much accelerated testing cycles. Further, its use of cloud tools also ensures greater scalability, and integration of it with CI/CD is smooth. But the method has some drawbacks such as the initial lack of designing an optimal POM structure and possible work increase for

maintenance when several abstraction layers are used. In this paper, Singh Dhaliwal et al. [2] presents the Hybrid Automation Framework that combines important test automation test independence, idempotency, and transparency principles enhancing software testing scalability and maintainability. The architecture is founded on a modular design and the POM pattern to enable flexibility across various test environments, with the major features including the Object Repository, Name Mapping, Functions Library, and Data Manager to enable easy running of tests and management. Among the greatest merits of this design is that it possesses the ability to support more than a single interface, such as the Command Line Interface (CLI), Application Programming Interface (API), and Graphical User Interface (GUI), thus offering flexibility in matching itself against diverse testing requirements. Again, with dynamic logging and reporting, it makes failure analysis more straightforward as well as enhances communication to the stakeholders. A major drawback, however, is the complexity of implementing and managing such an elaborate framework since handling several layers of automation was a very demanding task requiring much expertise and effort.Andrea Stocco et al. [3] suggest in their work a clustering-based technique for automated Page Object generation with the aim of minimizing the amount of manual effort involved in web testing. Their approach, realized in the APOGEN tool, programmatically discovers semantic abstractions from web pages and maps them as Java-based Page Objects. APOGEN compiles similar-looking elements across multiple web pages into reusable, abstract business functions, which can be called by test scripts in an automated test case. The second benefit is its potential to strongly decrease the human effort involved since 75% of the resultant code is utilized directly without amendment and 84% of generated methods correlate directly with meaningful behavior abstractions. Moreover, its clustering algorithm facilitates that generated Page Objects highly reflect the Page Objects developed by people, and its variations are kept minimal. But a limitation of this approach is its dependence on the clustering accuracy in case the web element is not correctly identified by the clustering process, it can result in incorrect or duplicate Page Objects that need manual corrections.

III. Proposed Advanced Page Object Model Design

To resolve the shortcomings of conventional POM frameworks, this research introduces an Advanced POM Design that combines modern software engineering principles to improve scalability, maintainability, and flexibility in test automation. The suggested approach unveils a standardized and module-like way to facilitate better organization of code, decreased redundancy, and easy support to UI modification. The advanced POM architecture takes into consideration multi-layer architecture, object initialization based on patterns utilizing factories, fluid interfaces to maximize script legibility, module-style design utilizing components, and externally configured data-based testing. All of these improvements cumulatively make available a robust and effective framework of automation strong enough to perform complicated web applications having dynamic UI elements. Flowchart representation of the Advanced POM Design is shown in fig. 1.

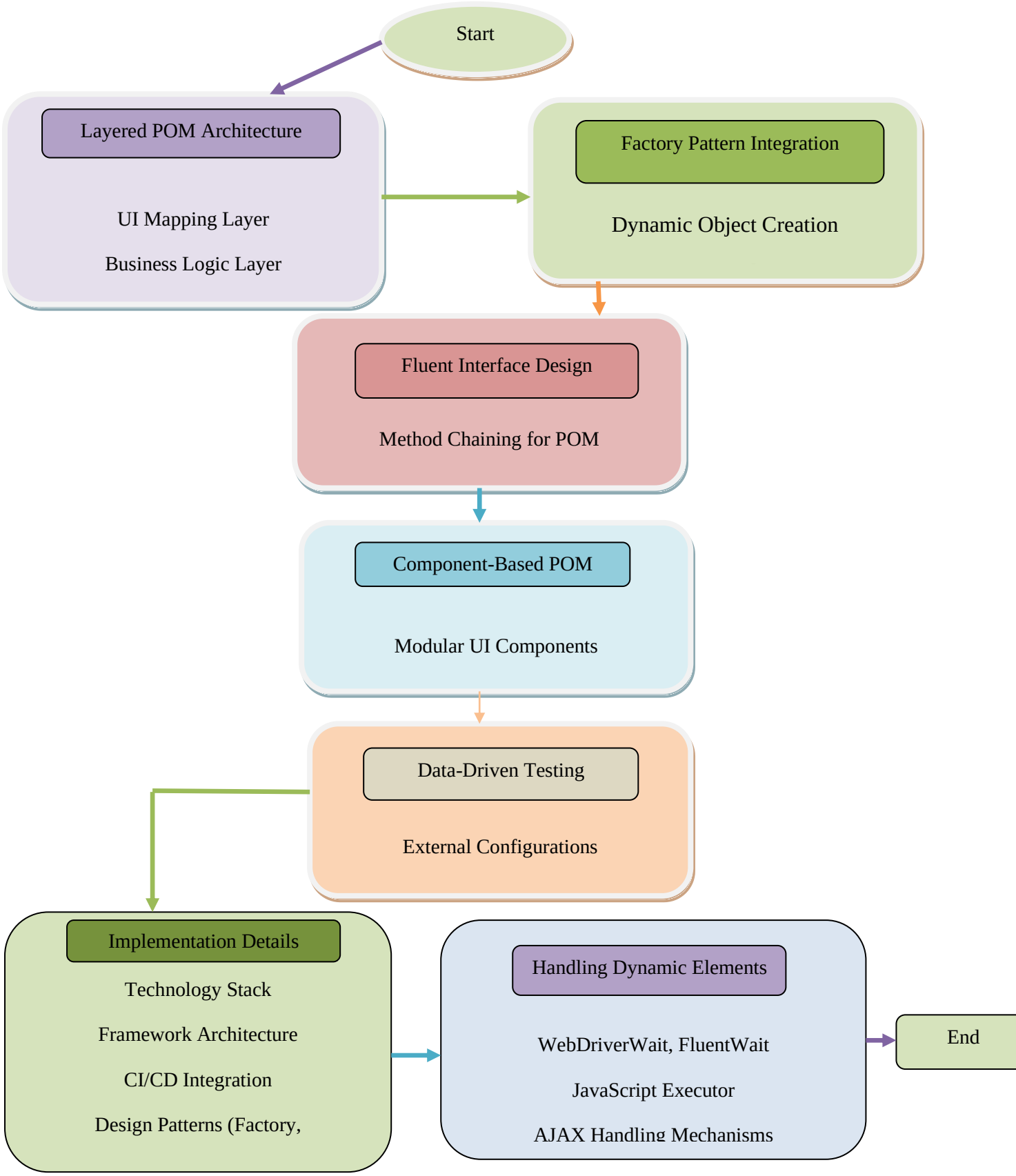
Start
Layered POM Architecture
UI Mapping Layer
Business Logic Layer
Factory Pattern Integration
Dynamic Object Creation
Fluent Interface Design
Method Chaining for POM
Component-Based POM
Modular UI Components
Data-Driven Testing
External Configurations
Implementation Details
Technology Stack
Framework Architecture
CI/CD Integration
Design Patterns (Factory,
Handling Dynamic Elements
WebDriverWait, FluentWait
JavaScript Executor
AJAX Handling Mechanisms
End

Fig. 1. Flowchart representation of the Advanced POM Design.

Important Improvements to the Proposed Advanced POM Design

A.Layered POM Architecture: Distinguishing Issues

A primary improvement with the proposed model is the incorporation of a multi-layered structure where various aspects of automation are demarcated as separate layers. This increases maintainability with separate demarcation of UI interaction, business rules, and execution of the tests, leaving behind less opportunity for code repetition and increasing the potential for reuse. The structure includes:

1)UI Mapping Layer

The UI Mapping Layer holds web elements, locators, and page structure representations centralized as an abstraction layer in between test scripts and the UI. It makes it simpler to maintain by having locators stored in a single location, reducing redundancy and simplifying updating when UI modifications are made. The layer simplifies scaling as well because it makes changes in element identifiers decouple from changing the test logic. It makes tests more resilient and stable to dynamic web applications through practices like Page Factory, dynamic element identification, and lazy initialization. Through separation of UI components from test scripts, the UI Mapping Layer ensures automation of maintenance and flexible, reliable test runs.

2)Business Logic Layer

Business Logic Layer is a key part of the Advanced POM that contains reusable interactions and methods to carry out specific actions on UI objects. It separates application-specific logic from test scripts and thus makes it more reusable, modular, and maintainable. Rather than receiving explicit UI interaction in test cases, high-level operations such as user authentication, form submission, or navigation flows are defined in the business logic layer, i.e., these operations can be reused in multiple test cases. For example, a login function would take care of typing credentials and clicking the login button, and thus test scripts would be able to call this procedure without dealing with specific UI elements. This method structures code better, eliminates duplication, and makes updates when application workflows have to change easier. It even encourages scalability because test scripts are neat, validation-logic based, and devoid of implied UI element changes, which enhances general test automation effectiveness.

3)Test Execution Layer

The Test Execution Layer carries out test cases by calling business logic methods as opposed to the direct manipulation of UI elements. It provides good separation of concerns, better readability, and maintainability. By concentrating on validation logic instead of UI interaction, test scripts are cleaner, modular, and maintainable. The differences in the design of UI do not impact test scripts, which makes it easier to maintain. This layer also facilitates test framework integration with TestNG, JUnit, or Cucumber for structured execution, parallelization, and fine-grained reporting for performance-driven automation.

B.Integration of Factory Patterns: Dynamic Object Generation

The Factory Pattern Integration within the POM Advanced enhances flexibility and extensibility by means of dynamic page object generation and management and eliminating the rigidity of the traditional

POM. Rather than explicitly creating instances of page classes within test code, a factory method automatizes page instantiation to cut boilerplate code and enhance maintainability. This method focuses on object creation, with the test automation platforms being capable of dealing with numerous implementations of a page class during runtime, which is very useful while working with various application states or UI changes. Since the objects' initialization is maintained within a single factory class, changes are simpler, with changing the pages' creation logic no longer implies doing it to numerous test scripts. This pattern provides more structured, tidier automation foundation, with increased code reuseability and less work on maintenance within large test frameworks.

Traditional approach:

```
LoginPage login = new LoginPage(driver);
login.enterUsername("user123");
login.enterPassword("pass123");
login.clickLogin();
```

Fluent Interface Approach:

```
new LoginPage(driver)
.enterUsername("user123")
.enterPassword("pass123")
.clickLogin();
```

1)Fluent Interface for Test Scripts: A Simplified Approach Chaining

The Fluent Interface Pattern to test automation adds readability and ease of maintenance on the scripts via supporting chaining on methods, producing resulting test cases that are easier to understand and intuitive. As opposed to the standard POM practice, where distinct method calls for every activity mean that the test steps have to be written as long and repeated scripts, fluent interface enables the test steps to be expressed in one logical flow. It is made possible by method design in a way that the page object instance is returned, making it possible to chain actions easily. Therefore, test scripts get reduced, organized, and readable with less unnecessary code and more readability of execution. Automation frameworks are served better through this method with fewer complexities in test logic, improved scalability, and improved maintainability, allowing for efficient test development and running.

C. Component-Based POM: Modular Reusability

Component-Based POM enhances test automation by designing reusable modular components rather than defining entire pages as single objects. It is suitable for repeated UI components such as navigation menus or search fields, avoiding duplication of code and easy maintenance. With these components divided into separate components, they would be reusable in various page objects with ease, ensuring consistency and scalability. This modularity also structures the automation frameworks to be more efficient and responsive, with ease of updates whenever the UI gets changed.

```
public class NavigationBar {
WebDriver driver;
    @FindBy(id = "home") WebElementhomeButton;
```

```
    @FindBy(id = "profile") WebElementprofileButton;
    public NavigationBar(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
}
public void navigateToProfile() {
profileButton.click();
}
}
```

D.Data-Driven POM with External Configurations:

The Data-Driven POM increases test automation by de-coupling test data from scripts and storing it in external resources like JSON, YAML, or Excel. The traditional POM is hardcoded-based, and modifications are challenging when test data is modified. With the test data stored in external resources, the automation engineers can modify input values without modifying the code, and hence test automation is made flexible and maintenance-friendly. This makes test scripts capable of fetching data dynamically, supporting multi-environment configurations and various test scenarios with ease. This reduces the script maintenance load, enhances reusability, and makes test execution simple, making the automation frameworks scalable and responsive to change.

Example: Fetching test data from a JSON file

```
{
    "username": "testUser",
    "password": "securePass123",
    "url": "https://example.com"
}
```

Test script dynamically loads data:

```
JSONObjecttestData = new JSONParser().parse(new FileReader("testData.json"));
String username = testData.get("username").toString();
String password = testData.get("password").toString();
loginPage.enterUsername(username).enterPassword(password).clickLogin();
```

E.Implementation Details

The application of an Advanced POM includes choosing the appropriate technology stack and suitable framework architecture for adequate automation.

1) Technology Stack

The automation framework can be developed using tools such as Selenium, Appium, Cypress, or Playwright, based on the type of application (web, mobile, or hybrid). These tools allow cross-browser and cross-platform testing, providing broad test coverage.

2) Framework Architecture

The automation framework uses a hierarchical folder structure, usually with folders for page objects, test scripts, utilities, config files, and reports. Code examples showing integration of external data sources, logging, and error handling make the code more maintainable. The framework should also be capable of supporting Continuous Integration/Continuous Deployment (CI/CD) pipelines so that automated tests can be run and reported in tools such as Jenkins, GitHub Actions, or GitLab CI/CD.

3) Design Patterns Used

Some design patterns boost the scalability, maintainability, and efficiency of a sophisticated POM framework. The Factory Pattern dynamically automates the creation of page objects and eliminates manual instantiation within test scripts to promote increased flexibility. The Singleton Pattern only maintains one active instance of the WebDriver throughout test execution to ensure efficient usage of resources and elimination of redundant browser sessions. Dependency Injection enhances modularity by introducing the necessary dependencies during runtime instead of hard-coding them, rendering the framework more flexible, reusable, and manageable. Coupled together, these design patterns optimize test automation to eliminate redundancy while enhancing the efficiency of the entire framework.

F.Handling Dynamic Elements and AJAX Calls:

Modern web applications tend to have dynamic aspects and AJAX calls, which require explicit waits and retry mechanisms to handle loading time. WebDriverWait (Selenium), Fluent Wait, or polling methods help to make the system stable by waiting for elements to be available before they are utilized. JavaScript Executor can be utilized for dealing with non-synchronously loaded

IV. RESULT AND DISCUSSION

To critically compare and contrast the performance of the Advanced POM with the Traditional POM, we will consider the most important comparison parameters like Maintainability, Code Reusability, Execution Time, and Scalability. The performance comparison will also involve experimental results-based benchmarking and real application-based benchmarking

A. Comparison Metrics

Maintainability: Maintainability refers to how easy it is to change or upgrade the test scripts when the application's business logic or UI is altered.

Code Reusability: Code reusability determines how well components like page objects and business logic can be reused across projects or test cases

Execution Time: Execution time refers to the amount of time it takes for a test case or set of test cases to execute and reflects how effective the framework is.

Scalability: Scalability defines how efficiently the framework can evolve with an increased number of test cases, increased application complexity, or increased users.

B. Benchmarking Against Traditional POM: Efficiency Improvements

Table I shows the Comparison of maintainability. The benchmarking results comparing the Traditional POM and the Advanced POM show substantial gains in efficiency, especially maintenance and code structure. In the Traditional POM, each change to the UI normally involves modifications to several test scripts, with 10 average code changes per change. This translates to a substantial amount of time spent on maintenance, up to 4 hours per UI change. Secondly, the code is repeatedly copied within different

Secondly, the code is repeated multiple times in multiple modules, and it has a redundancy of approximately 35%, which adds to the complexity and updation problems. This is as opposed to the Advanced POM framework, where locators and business logic are placed at one location and the number of modifications is only 2 per updation. This efficient mechanism saves maintenance time by far to a mere 1 hour per UI updation. The redundancy of the code is also minimized to a mere 10% for improved reusability and maintainability.

Moreover, the Advanced POM also increases ease of locator updates and decreases the number of modules to be updated from 4 in the conventional model to only 1, which makes the process faster and more efficient. This helps create a more scalable and flexible framework, which is suitable for contemporary web applications with regular UI updates. Graphical representation for the traditional POM vs advanced POM is mentioned in fig. 2.

TABLE I: COMPARISON OF MAINTAINABILITY

Metric	Traditional POM	Advanced POM
Number of Code Modifications for UI Changes	10 per change	2 per change
Time for UI Change Maintenance	4 hours	1 hour
Code Duplication (%)	35%	10%
Ease of Updating Locators	Moderate	Easy
Number of Modules Requiring Update	4	1

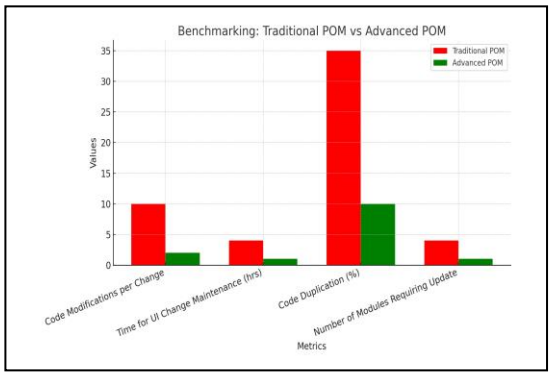

Fig. 2. Graphical representation for the traditional POM vs advanced POM.

Code Reusability

Table II depicts the Code Reusability. The results of Code Reusability benchmarking between the Traditional POM and the Advanced POM clearly show a large improvement in reusability of page objects and business logic. Page objects are moderately reusable in the Traditional POM, whereas the business logic is typically inlined directly in the test cases and cannot be reused for different tests. Therefore, there is low reusability of business logic. Also, introducing a new test case requires approximately 3 hours as it involves repeating existing methods and adjusting the page object logic in

each new scenario. Also, there are some 15 duplicated methods in the framework, increasing complexity and obstructing maintenance. In contrast, the Advanced POM approach strengthens both page object and business logic reusability by concentrating logic and building modular, reusable elements. This extremely high reusability enables new test cases to be incorporated significantly more effectively, bringing down the time needed from 3 hours to merely 1 hour. The count of copied methods also decreases from 15 to 5, making the framework more efficient and the code better organized. This leads to a more scalable and effective test automation framework with less redundancy and easier updates. Graphical representation for the code Reusability is mentioned in fig. 3.

TABLE II: CODE REUSABILITY

Metric	Traditional POM	Advanced POM
Reusability of Page Objects	Moderate	High
Reusability of Business Logic	Low	High
Time to Add a New Test Case	3 hours	1 hour
Number of Duplicated Methods	15	5

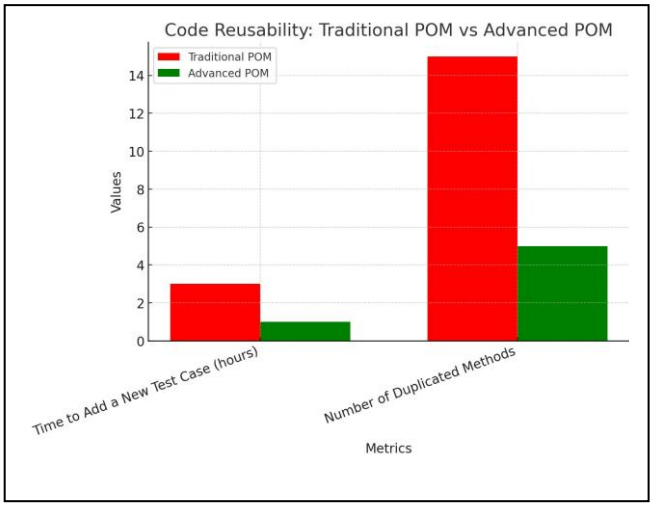

Fig. 3. Graphical representation for the code Reusability

Table III depicts the Execution Time Comparison. The Test Execution Time Comparison between the Classic POM and the Enhanced POM indicates a significant decrease in test run time, leading to faster and more efficient test cycles. Under the Classic POM, each test case takes approximately 15 minutes to run, mostly because of the instantiation of a new WebDriver object for every test within the suite. This leads to a total run time of 1500 minutes for 100 tests. Further, the overhead of initializing and maintaining multiple instances of WebDriver leads to inefficiencies in the test process. By contrast, the

Advanced POM framework improves upon this process by reusing a single instance of WebDriver throughout the test suite, with dramatic reduction in setup and teardown times. Consequently, each test case takes only 10 minutes to execute, and overall test execution time for 100 tests reduces to 1000 minutes. The test execution time is further decreased by the capability of the framework to execute every test in merely 2 minutes, as opposed to 3 minutes per test in the legacy model. By reducing duplicate instances of WebDriver and optimizing test run, the Advanced POM results in a faster and more streamlined testing process with greater efficiency and less time-to-market for app releases.

TABLE III: EXECUTION TIME COMPARISON

Metric	Traditional POM	Advanced POM
Test Case Execution Time	15 minutes	10 minutes
Total Test Execution Time (for 100 tests)	1500 minutes	1000 minutes
Number of WebDriver Instances Created	100 per test suite	1 per test suite
Execution Time per Test	3 minutes per test	2 minutes per test

Scalability Comparison

Table IV depicts the Scalability Comparison.The Scalability Comparison of the Classic POM vs. the Advanced POM shows the huge improvements made in managing big test suites and accommodating dynamic changes in the UI. In the Classic POM, running 500 tests requires a substantial 75 hours mainly because of increased code complexity as well as because the framework does not have any modularity. The system also takes 10 hours to include 50 new test cases due to the tight coupling between test scripts and UI interactions. The overall complexity of code in the original model is greater with 2000 lines of code, and it becomes more challenging to scale and maintain as the test suite increases. Conversely, the Advanced POM framework significantly minimizes test execution time, running the same 500 tests in only 50 hours, as a result of better code structuring, less redundancy, and more efficient test execution techniques. The time taken to incorporate 50 new test cases also decreases from 10 hours to merely 4 hours, as a result of the high reusability of page objects and business logic. The Advanced POM structure also decreases code complexity by nearly 40%, to 1200 lines of code, making it more efficient and scalable. Moreover, handling dynamic UI change is greatly supported in the Advanced POM because it incorporates methodologies such as dynamic object creation and locator centralization, which allow the framework to adapt quickly to evolving web applications. This renders the Advanced POM more appropriate for big-sized, dynamic, and complex test automation projects.

TABLE IV: SCALABILITY COMPARISON

Metric	Traditional POM	Advanced POM
Test Execution Speed (for 500 tests)	75 hours	50 hours
Time to Add 50	10 hours	4 hours

New Test Cases		
Code Complexity (Lines of Code)	2000	1200
Ability to Handle Dynamic UI Changes	Low	High

C. Case Study on Real-World Application:

Table V depicts the Case Study Results. The e-commerce website test automation case study showcases the obvious advantages of the Advanced POM compared to the Traditional POM. The Advanced POM minimizes test case run time (15 minutes vs. 25 minutes), UI change maintenance time (2 hours vs. 8 hours), and enhances code reusability through page object centralization. It also facilitates higher scalability with respect to the addition of new features and higher flexibility towards dynamic UI changes. In comparison, the Traditional POM is likely to be plagued by duplicate code, requires constant maintenance, and has lower scalability and flexibility. Overall, the Advanced POM provides a more effective, maintainable, and scalable approach towards web automation.

TABLE V: CASE STUDY RESULTS - E-COMMERCE AUTOMATION

Test scenario	**Traditional POM**	**Advanced POM**
Test case execution time	25 minutes	15 minutes
Maintenance time for UI charges	8 hours	2 hours
Code reusability	Moderate (duplicate logic)	High (centralized page objects)
Scalability for New Features	Low (requires script updates)	High (modular updates)
Adaptability to Dynamic UI Changes	Low (needs frequent updates)	High (handles dynamic elements)

V. CONCLUSION

In short, the Advanced POM proposed here is a far superior replacement for traditional test automation frameworks in that it incorporates modern software engineering principles such as layered architecture, dynamic object creation, fluent interfaces, and modular components. These aspects not only facilitate easier and less redundant maintenance of test scripts but also more flexible and scalable, particularly in high UI change frequency environments. In addition to the flexibility of the framework, data-driven testing and external configurations have been incorporated. The benchmarking results illustrate dramatic improvements in maintainability, code reusability, run time, and scalability to make the Advanced POM an extremely efficient and stable solution for complex, dynamic web applications. The technique enables faster, more efficient test automation that is easy to scale with the growing requirements for testing, thereby maximizing overall testing efficiency and resource utilization.

REFERENCES

[1] F.Ricca,A. Marche o, andA.Stocco,"AI-based Test Automation: A Grey Literature Analysis,"*2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*,pp. 263 270,2021.

[2] A. Singh Dhaliwal,"Managing Artifacts and Binaries in Continuous Integration / Continuous Deployment (CI/CD) Pipelines,"*International Journal of Science and Research (IJSR)*, vol. 11, no. 6, pp. 2003–2005,2022. hps://doi.org/10.21275/SR24506190429.

[3] A.Stocco, M. Leotta, F.Ricca, and P. Tonella,"Clustering-aided page object generation for web testing,"*In Web Engineering: 16th International Conference, ICWE 2022, Lugano, Switzerland, June 6-9, 2022. Proceedings Springer International Publishing*, vol.16, pp. 132-151, 2022.

[4] J.Huoponen, "Designing a highly available and scalable cloud architecture for a web application," 2022.

[5] G. R.Mattiello, and A. T. Endo, "Model-based testing leveraged for automated web tests,"*Software Quality Journal*, vol. 30, no. 3, pp. 621-649,2022.

[6] A. Ali, "Enhancing Selenium Automation with Custom Frameworks".

[7] A.Kumar, S. K.Pandey,S.Prakash, K. U. Singh, T. Singh, and G. Kumar,"Enhancing web application efficiency: Exploring modern design patterns within the MVC framework,"*In 2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES) IEEE*, pp. 43-48, 2023, April.

[8] M. Leotta, B. García, F. Ricca, and J. Whitehead, "Challenges of end-to-end testing with selenium WebDriver and how to face them: A survey,"*In 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE*,pp. 339-350, 2023, April.

[9] M. Cunningham, N. Nigam, J. Mukhopadhaya, J. J. Alonso, and S. Ayyalasomayajula,"Multi-Fidelity Probabilistic Aerodynamic Database Generation with the ProForMA Tool,"*AIAA SCITECH 2023 Forum. AIAA SCITECH 2023 Forum, National Harbor, MD & Online*,2023, January 23. hps://doi.org/10.2514/6.2023 0653 14.

[10] V.Yadav, R. K.Botchway, R.Senkerik, andZ.KominkovaOplatkova,"Robotic Automation of Software Testing From a Machine,"2021. hps://doi.org/10.13164/mendel.2021.2.068

[11] S. Fatima, B. Mansoor, L. Ovais, S. A.Sadruddin, and S. A. Hashmi, "Automated Testing with Machine Learning Frameworks: A Critical Analysis,"*The 7th International Electrical Engineering Conference*,vol. 12,2022. hps://doi.org/10.3390/engproc2022020012

[12] B. García, M. Munoz-Organero, C. Alario-Hoyos, and C. D. Kloos,"Automated driver management for Selenium WebDriver,"*Empirical Software Engineering*,vol. 26, no. 5, pp. 107, 2021. hps://doi.org/10.1007/s10664-021 09975-3

[13] P.Ji, Y.Feng, J.Liu, Z.Zhao, andB. Xu,"Automated Testing for Machine Translation via Constituency Invariance,"*2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*,pp. 468–479,2021. h ps://doi.org/10.1109/ASE51524.2021.9678715

[14] S.Thummalapenta, S.Sinha, N. Singhania, andS. Chandra,"Automating test automation,"*2012 34th International Conference*,2012. hps://doi.org/10.1109/ICSE.2012.6227131.

[15] K. R.Halani, Kavita, and R. Saxena,"Critical Analysis of Manual Versus Automation Testing,"*2021 International Conference on Computational Performance Evaluation (ComPE)*,pp. 132–135,2021. hps://doi.org/10.1109/ComPE53109.2021.9752388

[16] J. Jain, "Tools, Frameworks, and Libraries,"*In J. Jain, Learn API Testing*, pp. 41–73, 2022. Apress.h ps://doi.org/10.1007/978-1-4842-8142-0_4 21.P.Ji,Y.Feng, J.Liu, Z.Zhao, andB. Xu, "Automated Testing for Machine Translation via Constituency Invariance,"*2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*,pp. 468–479,2021. hps://doi.org/10.1109/ASE51524.2021.9678715

[17] M.Leotta,A.Stocco, F. Ricca, and P. Tonella,"Using multi-locators to increase the robustness of web test cases,"*In Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, IEEE*, pp. 1–10, 2022.

[18] M. Leotta, A. Stocco, F. Ricca, and P. Tonella,"ROBULA+: An algorithm for generating robustXPath locators for web testing,"*Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 177 204, 2016.

[19] V. Levenshtein,"Binary Codes Capable of Correcting Deletions, Insertions and Reversals,"*Soviet Physics Doklady*, vol. 10, pp. 707, 1966.

[20] S. Sampath,"Advances in user-session-based testing of web applications,"*Advances in Com puters*, vol. 86, pp. 87–108, 2012.