Challenges in GUI Test Automation for Dynamic Web Applications: A Systematic Review

Article · August 2025

CITATIONS
0

READS
32

2 authors, including:

Ali Abdalftah Fadul
Nile Valley University
10 PUBLICATIONS **2** CITATIONS

SEE PROFILE

Excellence Journal for Engineering Sciences
Volume 2, Issue 5, Aug 2025–ISSN 1858–9448
journals.eaeu.edu.sd

Challenges in GUI Test Automation for Dynamic Web Applications: A Systematic Review

Ali Abdalftah Fadul Mohammed[1] Prof. Khaled Ahmed Ibrahim[2]
1 Faculty of Computer Sciences & Info. Technology, Nile Valley University, Elsheikh Abdallah Elbadri University
2 Faculty of Computer Sciences & Info. Technology, Karary University
Email: aliftah@gmail.com, aliftah@nilevalley.edu.sd[1]; kalidaik123@gmail.com[2]

Received: 9. July.2025 Accepted :30. July. 2025 Published: 12. Aug. 2025

Abstract
Automated testing of Graphical User Interfaces (GUIs) plays a critical role in ensuring the reliability and functionality of dynamic web applications. However, GUI testing remains challenging due to frequent UI changes, synchronization issues, and the complexity of modern web environments. This paper presents a systematic review of recent academic studies (2021–2025) that investigate the key challenges and proposed solutions in GUI automation testing for dynamic web applications. A total of 17 peer reviewed studies were selected using a structured literature search and screening process based on predefined inclusion criteria. The methodology followed the principles of systematic review, including study selection, data extraction, and thematic analysis. Each study was analyzed in terms of its objectives, identified challenges, applied techniques, and reported outcomes. The review reveals that common challenges include high maintenance costs of test scripts, poor script reusability, difficulty handling dynamic GUI elements, and the need for significant technical expertise. Proposed solutions range from hybrid testing frameworks (combining keyword driven, data driven, and modular approaches) to the application of AI based techniques. Despite these efforts, limitations persist, especially in scalability and handling frequent UI changes. The main contribution of this paper lies in synthesizing current research trends and highlighting key areas that need further investigation. It also provides recommendations for researchers and practitioners to improve the design of automation frameworks, particularly for dynamic and frequently updated web systems. This review serves as a foundation for future work toward more intelligent and adaptable GUI testing solutions.

Keywords: *Automated Software Testing, GUI Automation Testing, Dynamic Web Applications, Test Automation Challenges, Web UI Testing*

I. INTRODUCTION

Graphical User Interface (GUI) automation testing has become a cornerstone in contemporary software development and quality assurance practices. With the growing demand for interactive and user friendly web applications, ensuring the reliability, functionality, and performance of user interfaces is more critical than ever. Automated testing provides an efficient mechanism to validate application behavior across various platforms and configurations, reducing the reliance on manual efforts and enhancing the consistency of test outcomes. In a development landscape increasingly defined by speed, scalability, and continuous delivery, GUI automation testing plays a crucial role in maintaining high software standards and supporting rapid release cycles.

However, the rise of dynamic web applications has fundamentally transformed the nature of GUI testing. Unlike static web pages, dynamic web applications

depend heavily on client side rendering technologies, asynchronous data exchanges, and real time user interactions. These features, while beneficial to the user experience, pose complex technical challenges for test automation frameworks. Traditional testing strategies often struggle to cope with these environments, where interface elements frequently change, load at unpredictable times, or vary depending on user behavior and application state. As a result, test scripts become fragile, locators become unreliable, and synchronization becomes difficult to manage.

Such complexities have rendered many existing automation tools and techniques insufficient. The frequent modification of Document Object Model (DOM) structures causes test cases to fail, not due to actual defects in the application, but because the testing scripts cannot locate or interact with the dynamic elements reliably. This phenomenon, commonly referred to as test flakiness, undermines the credibility of automated testing and creates additional burdens in test maintenance and debugging. Furthermore, challenges such as cross browser compatibility, high test execution time, and the steep learning curve of automation tools continue to limit the adoption and effectiveness of automation in real world scenarios.

In response to these obstacles, the research and industry communities have proposed various solutions aimed at improving test reliability and adaptability. Approaches such as the Page Object Model (POM) have been introduced to enhance code reusability and structure, while intelligent locators and self-healing techniques have emerged to address element identification issues. Tools that support scriptless or low code testing environments are gaining popularity, especially among teams with limited programming expertise. Additionally, integrating automation tools into CI/CD pipelines has become a standard practice to ensure continuous and timely feedback during development.

Despite these advancements, a unified and adaptive solution that effectively handles all aspects of dynamic web application testing remains elusive. Many of the proposed techniques offer partial improvements but fail to address the root causes of automation fragility in dynamic environments. The complexity of modern applications, combined with the evolving nature of web technologies, calls for a deeper understanding of the challenges at hand and a systematic exploration of available solutions.

This paper presents a systematic review of recent studies focusing on GUI automation testing for dynamic web applications. By examining research published between 2021 and 2025, this study aims to consolidate current knowledge in the field, identify common testing challenges, and evaluate the practical and theoretical effectiveness of proposed approaches. The review also seeks to uncover limitations in current tools and frameworks, explore emerging trends such as AI driven testing, and provide informed insights that can guide future innovation in automated GUI testing. In doing so, this work contributes to the ongoing effort to build more intelligent, scalable, and resilient automation solutions suited to the rapidly evolving landscape of dynamic web applications.

II. STUDY SELECTION AND METHODOLOGY

This review employed a structured methodology to identify, evaluate, and synthesize relevant research on GUI automation testing challenges for dynamic web applications. The process adhered to the following steps:

1.Timeframe & Scope:

Studies published between January 2021 and March 2025 were prioritized to capture the most recent advancements and challenges in rapidly evolving web technologies.

2.Search Strategy:

Systematic searches were conducted across major academic databases, including:

- IEEE Xplore
- ACM Digital Library (ACM DL)
- ScienceDirect
- Springer Link
- Google Scholar

3.Keywords:

Primary search terms included combinations of:
"GUI Automation Testing", "Web UI Testing", "Automated GUI Testing" "Dynamic Web Applications", "Dynamic UI Testing" "Test Automation Challenges", "Page Object Model", "Selenium", "Playwright", "Scriptless Testing".
Boolean operators (AND, OR) and wildcards (*) were used to refine results.

4.Inclusion Criteria:

- Focus on empirical studies, case studies, or systematic reviews related to GUI automation challenges.
- Emphasis on dynamic web applications.
- Solutions/tools evaluated in industrial or realistic academic settings.
- Peer reviewed publications or reputable conference proceedings.

5.Exclusion Criteria:

- Studies predating 2021 (unless foundational).
- Non-English publications.
- Articles focusing solely on non-web GUI testing.

6.Screening & Selection Process:

The systematic review employed a three phase selection process: initial screening involved evaluating titles and abstracts of approximately 67 identified studies against predefined inclusion criteria; this was followed by a comprehensive full text review of 34 studies that passed the initial screening; ultimately, 17 studies that met all inclusion criteria were selected for final synthesis and analysis.

III. LITERATURE REVIEW

(Fadul & Ibrahim, 2025) Addresses key challenges in GUI test automation for dynamic web applications, including test instability due to changing element locators, inadequate input scenario coverage, and undetected paths from dynamic interactions. To overcome these, the authors propose an integrated framework combining DOM analysis for precise element identification, the Page Object Model (POM) to decouple elements from test scripts (stored in reusable JSON files), and automated random test data generation (via PHP/JSON) to broaden input coverage. Methodologically, they implemented a 7 page PHP/JavaScript web application using Selenium WebDriver, featuring dedicated modules for automated element extraction, data generation, and test execution with coverage metrics. Key results demonstrated a 39.3% improvement in execution efficiency using POM versus traditional methods, achieved 100% GUI element coverage via DOM parsing, and enhanced maintainability through centralized JSON management. Limitations included JSON's reduced readability for complex data (partially mitigated by a custom PHP UI) and untested scalability in highly complex applications like mobile environments.

(Neelapu, 2025) Examines hybrid testing frameworks for GUI automation, addressing key challenges such as high maintenance costs due to dynamic UI changes, limited reusability of test scripts (especially in linear and module based frameworks), technical expertise requirements for framework setup, and scalability issues with complex web applications (e.g., dynamic pages). To mitigate these, the proposed hybrid methodology integrates keyword driven, data driven, modular, and linear scripting approaches within a multilayered architecture, separating UI components from test logic to enhance adaptability. Solutions include reusable libraries for rapid UI change recovery, standardized coding practices, and cloud integration for scalability. Results confirm the framework's effectiveness in reducing manual effort, improving test coverage, ensuring cross platform consistency, and accelerating defect detection. However, limitations include initial setup complexity, dependency on skilled automation testers, and insufficient exploration of AI/ML driven testing for dynamic content.

(Jayaraman, 2025) Explores challenges in GUI test automation for dynamic web applications using Selenium and NUnit, identifying key hurdles including handling dynamic elements (AJAX, JavaScript), synchronization issues (element loading timing), cross browser compatibility, test script maintenance during UI changes, and test data management. To address these, the authors propose solutions such as explicit/implicit waits for synchronization, the Page Object Model (POM) to enhance maintainability, data driven testing via NUnit parameterization, and Selenium Grid for parallel cross browser execution. Methodologically, they implement a real world case study integrating Selenium WebDriver with NUnit, designing test scenarios (e.g., dynamic content, forms) while incorporating CI/CD pipelines and POM patterns. Key results demonstrate improved test coverage, faster regression cycles, and consistent cross browser behavior, though limitations include ongoing maintenance burdens for rapidly evolving UIs and scalability challenges with large parallel test suites.

Based on the comparative study by (Fadul & Ibrahim, 2024), test automation is critically important for dynamic web systems because it addresses their inherent complexity and volatility, such as frequent UI changes, dynamic content (e.g., AJAX, SPAs), and cross browser/device compatibility requirements. Automation enables efficient handling of these challenges through features like AI powered visual validation (e.g., Applitools) for detecting visual regressions, robust object identification/wait mechanisms (e.g., Selenium, Playwright) for interacting with unpredictable elements, and seamless cross environment testing to ensure consistent functionality. This significantly reduces flaky tests, maintenance effort, and human error compared to manual testing, while facilitating rapid regression testing, continuous integration (CI/CD), and scalable test coverage for complex user journeys—ultimately accelerating release cycles, lowering long term costs, and ensuring reliability in ever evolving web applications.

Based on the study by (Mothey, 2024), challenges in GUI test automation for dynamic web applications include handling frequent UI changes, cross browser compatibility, dynamic elements (e.g., wait conditions, flash objects), and the complexity of maintaining scripts for data centric applications like e-commerce platforms. Solutions proposed involve a data driven framework leveraging Selenium WebDriver, where test scenarios are managed via Excel spreadsheets to separate test data from scripts, enhancing reusability and reducing maintenance. Key results demonstrated improved test coverage, efficient handling of large datasets, and faster execution through parallelization. However, limitations included dependency on technical expertise for script maintenance, limited reusability validation of library functions, and scalability issues with extremely large test suites.

(Mahdy & Atef Raslan, 2024) Propose an automation testing framework for educational web applications using Robot Framework and Selenium, adopting a Page Object Model (POM) and keyword driven testing approach to streamline test case development and maintenance. The research addresses challenges such as the complexity of testing dynamic web applications due to multi platform environments, sensitivity to changing UI elements, and difficulties in maintaining test scripts during application updates. To overcome these, the researchers suggest solutions like leveraging POM for code reusability, integrating Selenium to handle dynamic elements, and implementing sandbox based security policies. The methodology involves generating POM elements from web applications, creating test scripts, executing them via Robot Framework, and logging results for analysis. Key findings include the framework's effectiveness in improving testing efficiency and supporting multi specification execution, particularly for educational systems. However, the study faced limitations such as initial setup complexity and integration challenges with external tools.

(CAMARGO, 2024) Addresses challenges in GUI automation testing, such as dynamic web environments, multi platform compatibility issues, and difficulties in generalizing tests for diverse web platforms. It proposes solutions like leveraging Selenium for handling dynamic elements, adopting Page Object Model (POM) for code reusability, and integrating behavior driven development (BDD) tools like Cucumber and Katalon. The methodology employs a systematic mapping study (SMS), analyzing 37 peer reviewed studies (2019–2023) using inclusion/exclusion criteria, with a focus on tools, techniques, and limitations in GUI automation. Key findings reveal that 64.8% of studies proposed novel tools or methodologies, emphasizing model based test case generation, reinforcement learning, and image based GUI detection. However, limitations include high implementation costs, scalability issues for cross platform testing, and challenges in adapting to rapidly evolving UIs.

(Mathew, 2024) Addresses challenges such as dynamic UI elements, cross browser compatibility, maintenance of test scripts during frequent updates, and high initial implementation costs in GUI automation. To mitigate these, the study proposes Selenium WebDriver for dynamic element handling, Page Object Model (POM) for modular test design, and integration with CI/CD pipelines for continuous testing. The methodology employs a case study approach, analyzing a real world dynamic web application to evaluate the effectiveness of the proposed framework through test script

development, execution, and defect tracking. Key findings demonstrate that the framework significantly improves test stability and reusability, reducing manual intervention by 40% while maintaining high test coverage. However, limitations include complexity in configuring dynamic locators and scalability issues in large scale applications. This study is highly relevant to the PhD research on "Challenges in GUI Automation Testing: A Case Study on Dynamic Web Application", as it provides empirical validation of automation strategies for dynamic environments, though further exploration into AI driven test generation and adaptive CI/CD integration is recommended to address evolving UI complexities.

(Nass, 2024) Addresses persistent challenges in GUI based test automation, particularly fragile test execution, high maintenance costs, and the demand for programming skills. Key challenges identified via a systematic literature review include: non-robust GUI widget identification, synchronization issues between tests and the system under test (SUT), high skill requirements for automation, and difficulties in creating/maintaining model based tests. To mitigate these, the study proposes two solutions: Augmented Testing (AT), implemented via the Scout prototype, which reduces skill dependencies by superimposing test guidance on the GUI; and Similo (later enhanced to VON Similo and VON Similo LLM), a similarity based web element localization approach that improves robustness by using weighted similarity scores across multiple element properties. The methodology combines a multi method approach: a longitudinal systematic review, industrial workshops (e.g., with Ericsson and Inceptive), quasi experiments, and empirical evaluations on real world web applications. Key results indicate that Scout reduced test creation time by 50–90%, while Similo variants increased localization accuracy by 10–44% over baseline methods.

(MOURA, 2024) Addresses the critical challenges of scriptless GUI testing for web applications, specifically focusing on the unique discovery of actionable elements, robust synchronization with the application under test (AUT), and efficient systematic exploration to manage prolonged execution times in industrial settings. To tackle these issues, the study introduces and evaluates three key solutions integrated into the Cytestion tool: the Unique Actionable Elements Search (UAES) approach for non-redundant element discovery, the Network Wait mechanism for improved synchronization, and the Iterative Deepening URL Based Search (IDUBS) algorithm for efficient exploration. The methodology involved developing these techniques, incorporating them into Cytestion, and conducting empirical evaluations comparing Cytestion's performance against a state of the art GUI testing tool (TESTAR) using both open source and industrial web applications, measuring fault detection and runtime efficiency. The main findings demonstrate that Cytestion, leveraging UAES, Network Wait, and IDUBS, significantly outperforms the baseline tool in detecting visible failures and reducing execution time.

(Manukonda, 2024) Addresses challenges in GUI test automation, particularly the inefficiency and limited coverage of cross browser/OS testing within Agile's rapid cycles. It proposes Selenium Grid's distributed testing architecture as the key solution, utilizing a hub node model to enable parallel test execution across diverse environments. The methodology involved implementing and empirically evaluating Selenium Grid in Agile settings, measuring its impact on test coverage and execution speed. Key results demonstrated significant improvements in test automation coverage (through broader configuration testing) and efficiency (via reduced execution time), aligning with Agile's need for rapid feedback. However, the study notes limitations, including potential performance bottlenecks in the Grid itself and the complexity of managing distributed node configurations.

(Inturi, 2023) Addresses challenges in GUI test automation, particularly dynamic element identification (e.g., locating web elements via XPath/IDs prone to failure with UI changes) and test fragility due to frequent interface updates. Solutions proposed include adopting the Page Object Model (POM) design pattern to decouple test scripts from UI elements—enhancing maintainability—and leveraging TestNG for structured reporting and parallel execution. The methodology employed a descriptive, scenario based observational approach, implementing a Selenium WebDriver framework

(with Java, Maven, and TestNG) to automate functional testing of NanoHub.org, a dynamic web application for nanoscale simulations. Key results confirmed significant time reduction in regression testing cycles but revealed reliability issues (e.g., 35% of test failures stemmed from non-code factors like environment instability). Study limitations included potential error inflation in time saving metrics, restricted cross browser validation (Chrome only focus), and synchronization challenges with dynamic content .

(Björkman, 2023) Investigates GUI test automation challenges and solutions for a dynamic e-learning web application using Playwright. Key challenges included dynamic element loading causing synchronization issues, flaky tests due to race conditions and backend limitations, non-resettable test environments hindering complex user flows (e.g., teacher student interactions), and fragile locators (CSS/ID-based) prone to UI changes. Solutions involved leveraging Playwright's auto wait/retry features for dynamic elements, serial test execution with timeouts to avoid concurrency conflicts, user facing locators (e.g., getByRole()), and failure screenshots for debugging. The methodology followed Design Science Research (DSRM): requirements gathering via stakeholder interviews, comparative analysis of tools (selecting Playwright for JavaScript support and robustness). Key findings confirmed Playwright's low learning curve but highlighted critical dependencies: test environment controllability (data reset), close developer tester collaboration to address backend constraints, and flaky tests' erosion of trust in automation.

(Sezer, 2023) Investigates challenges and evaluation methods for GUI test automation, particularly within constrained environments where fault injection or traditional case studies are infeasible. Key challenges identified include the test oracle problem (difficulty automating result verification), high maintenance costs for test scripts due to frequent GUI changes (especially in script based approaches like Capture & Replay and code based testing), long execution times in scriptless techniques, reproducibility issues with failures found by random tools, cross platform complexity, and the need for specialized technical knowledge for tools like Tosca. To address the core challenge of evaluating techniques without fault injection, this study designed a framework through a quasi experiment that compares a script based tool (Tosca, model based) with a scriptless tool (Testar, random testing) on a web application (Tricentis Demo Web Shop). Key findings revealed that Tosca detected more functional bugs (11 high severity functional issues) but required significant time for test design, training (up to 18 hours), and maintenance. Testar excelled at finding system level crashes/freezes (4 high severity system failures) with lower setup/execution effort but suffered from limited failure reproducibility and redundant detections (9 of 16 initial findings were duplicates/non-bugs). Efficiency metrics favored Testar for execution, while Tosca incurred high overhead in design and analysis.

(K.JAGANESHWARI & S.DJODILATCHOUMY, 2022) Addresses challenges in GUI automation testing, including high maintenance costs due to dynamic web element changes, dependency on large labeled training datasets, and resource intensive manual test case creation. Their solution integrates computer vision (OpenCV) and machine learning (DNNs) with Selenium WebDriver to automate widget detection/classification via exploratory behavioral analysis (EBA), eliminating the need for extensive repositories by using URL based screenshots and blob detection for real time element identification. The methodology involves: Capturing screenshots using PyAutoGUI; Detecting widgets via BLOB text detection and bounding boxes; and classifying elements (e.g., buttons, input fields) using DNNs; Key results showed 98.4% test coverage accuracy, reduced testing time (~10 seconds/page), and lower maintenance costs compared to existing tools, with a 35% reduction in RMSE error rates. Limitations included reliance on a single dataset (Frozen EAST) and failure to achieve 100% accuracy due to complex GUI variations.

(Lahtinen, 2022) Investigates challenges in automating GUI regression testing for a dynamic web application in the electricity trading sector, employing a design science methodology involving iterative problem identification, requirement gathering from stakeholders, and artifact development. Key GUI automation challenges included frequent UI changes causing test brittleness,

difficulties in maintaining element locators (especially with ASP.NET-generated attributes), complex integrations with third party services, asynchronous behavior leading to flaky tests, and resource constraints in small companies. To address these, the study proposed solutions centered on Selenium WebDriver integrated with xUnit, utilizing CSS selectors over XPath for maintainability, implementing a shared function library to reduce code duplication, and establishing a recording to code pipeline via Selenium IDE for initial test prototyping by non-technical testers. The main results demonstrated a successful CI/CD pipeline integration in Azure DevOps, reducing manual regression efforts for critical paths, though test coverage remained limited. Study limitations included scarce resources hindering comprehensive implementation, insufficient database testing due to architectural constraints, and challenges in dynamic element handling.

(Thooriqoh, Tiara Nur Annisa, & Umi Laili Yuhana, 2021) Examined Selenium frameworks for web automation testing, identifying key challenges in GUI automation testing such as test script fragility due to UI changes, high resource consumption, limited built-in reporting, and difficulties in testing dynamic web elements. Solutions proposed included adopting intelligent locator strategies (e.g., SmartDriver's self-healing for moved elements), using headless browsers to reduce resource overhead (20%+ efficiency gain), integrating AI/ML for predictive test maintenance (e.g., SVM classifiers for element specific test cases), and combining Selenium with tools like FitNesse for modular frameworks. Key findings confirmed Selenium's flexibility for functional, security (e.g., XSS vulnerability testing), and AI augmented testing, but highlighted limitations like dependency on third party tools for reporting, scalability issues in dynamic applications, and insufficient focus on long term maintainability.

(Nass, Emil Alégroth, & Robert Feldt, 2021) Examines the persistent challenges in automating Graphical User Interface (GUI) testing and explores the reasons why some of these challenges remain unsolved. The core issue addressed in the study is that GUI test automation continues to face technical difficulties despite advancements in tools and technologies, limiting its effectiveness compared to other automated testing methods such as unit and integration testing. The study aims to identify the key technical challenges hindering the progress of GUI test automation and to understand why they have persisted for so long without effective solutions. The research is based on a systematic literature review of 49 published studies, which were analyzed using thematic coding to identify 24 major challenges related to GUI test automation. Among the most common challenges identified are test execution failures due to application changes, difficulties in synchronizing tests with the application's state, and challenges in accurately recognizing GUI elements. The study also found that some of these challenges are inherent and cannot be entirely eliminated, while others can be addressed through improvements in tools and technologies. One of the study's limitations is that many documented challenges may not be sufficiently specific to provide direct solutions. Additionally, some proposed technical solutions have not been widely adopted in industrial practice.

IV. Comparative Analysis of Prior Research on GUI Automation Testing Challenges

Table 1 presents a synthesized comparative summary of seventeen notable research studies that examine GUI automation testing in dynamic web applications. These studies, published between 2021 and 2025, highlight diverse testing challenges and proposed solutions aimed at improving the robustness and reliability of GUI testing in modern web systems.

A common theme that emerges across these studies is the difficulty of managing test stability in response to frequent UI changes. Unlike static interfaces, dynamic web applications generate and manipulate interface elements in real time. As a result, DOM structures are highly volatile, which leads to frequent breakage of test scripts when locators fail to identify shifting or disappearing elements. This instability significantly increases the cost of maintaining automated test suites and undermines trust in the automation process.

Synchronization issues are also widely acknowledged as a persistent challenge. Many studies report problems arising from asynchronous operations, such as delayed loading of UI elements,

background API calls, or the dynamic rendering of components. When test scripts attempt to interact with elements that have not yet been loaded or initialized, execution errors and test failures are inevitable. To mitigate these risks, several research efforts have explored the implementation of explicit and implicit wait mechanisms, as well as advanced synchronization strategies that dynamically adapt to runtime conditions.

In addition to these technical challenges, scalability concerns emerge prominently in studies focusing on enterprise level or Agile driven development environments. Testing large scale applications across multiple platforms and browsers introduces substantial resource demands and execution time constraints. Without distributed execution capabilities or parallelization frameworks, such as those offered by Selenium Grid, the feasibility of maintaining comprehensive and timely test coverage is significantly reduced. Furthermore, scalability is not merely a matter of infrastructure it also involves designing modular, maintainable, and reusable test suites that can evolve alongside the application under test.

To address these multifaceted challenges, the reviewed studies propose a wide variety of solutions. Technical frameworks like Selenium WebDriver and Playwright are frequently used due to their flexibility, open source nature, and support for dynamic interaction models. Robot Framework, often used in combination with keyword driven testing, has also gained attention for its ease of use and ability to separate test logic from implementation. These tools are often paired with design patterns such as the Page Object Model (POM), which enhances modularity and improves the maintainability of test scripts over time.

Beyond conventional tools and patterns, several studies investigate the potential of advanced automation enhancements. These include the use of artificial intelligence and machine learning techniques to implement self-healing locators, predictive test case maintenance, and automated test generation. Smart locator strategies, such as similarity based or behavior driven element recognition, have shown promise in improving resilience to UI changes. Additionally, scriptless testing tools and visual testing platforms aim to reduce the need for programming expertise and make automation accessible to a broader range of users.

Despite the evident progress in tool capabilities and testing methodologies, the findings also highlight a number of recurring limitations. High setup complexity remains a barrier to adoption, particularly for small teams or resource constrained environments. Many automation solutions require significant configuration, infrastructure, and technical skill to implement effectively. Furthermore, while some tools offer scalability through distributed testing, others fall short in maintaining performance under high execution loads or diverse system configurations. The reliance on technical expertise also limits the involvement of non-developers in test creation and analysis, which runs counter to the collaborative ideals of Agile and DevOps practices.

Several studies emphasize the need for more adaptive and intelligent testing strategies that can autonomously respond to the everchanging nature of dynamic web applications. The integration of AI/ML driven automation, along with tighter coupling between test tools and CI/CD pipelines, is identified as a promising direction for future research and development. These approaches co uld help bridge the gap between application evolution and test maintenance, ultimately enabling more reliable, efficient, and sustainable testing processes.

In conclusion, the comparative analysis summarized in Table 1 offers valuable insights for both researchers and practitioners. It not only synthesizes the current state of GUI test automation in dynamic web contexts but also reveals areas where further innovation is needed. By examining the technical challenges, solution patterns, tool capabilities, and identified limitations, this review provides a foundational reference for advancing the effectiveness and adaptability of GUI testing in the modern web development landscape.

Table 1: Summary of Studies on GUI Automation Testing Challenges and Solutions for Dynamic Web Applications

Author(s) & Year	Key Challenges	Core Solutions	Key Results & Limitations
Fadul & Ibrahim (2025)	Unstable locators, insufficient input coverage, hidden dynamic paths	DOM parsing, POM with JSON storage, automated input generation	39.3% efficiency gain, full element coverage; JSON complexity, untested on mobile
Neelapu (2025)	High maintenance, limited script reusability, scalability issues, complex setup	Hybrid framework (keyword/data/modular), reusable libraries, cloud integration	Improved coverage and scalability; complex setup, lacks AI/ML integration
Jayaraman (2025)	Dynamic elements, synchronization, cross-browser issues, script maintenance	Selenium + NUnit, waits, POM, CI/CD, Selenium Grid	Better coverage and regression speed; scalability issues in large test suites
Mothey (2024)	Frequent UI changes, dynamic content, browser compatibility	Data-driven testing using Excel, parallel execution	Broad test coverage, faster execution; technical expertise required, reusability gaps
Mahdy & Raslan (2024)	Multi-platform support, UI sensitivity, script maintenance	Robot Framework + Selenium, keyword-driven, POM, sandbox policies	Increased efficiency and flexibility; tool integration complexity
CAMARGO (2024)	Cross-platform issues, evolving UIs, high implementation cost	POM, BDD (Cucumber/Katalon), SMS review of tools	64.8% introduced new methods; cost and scalability remain challenges
Mathew (2024)	Dynamic elements, cross-browser issues, script upkeep, high cost	Selenium WebDriver, POM, CI/CD pipelines	40% reduction in manual effort, stable tests; dynamic locator configuration is complex
Nass (2024)	Fragile execution, high skill requirements, sync and locator issues	Scout (Augmented Testing), Similo (similarity-based locator)	50–90% faster test creation, 10–44% improvement in locator accuracy
MOURA (2024)	Element discovery, synchronization, execution time	UAES, Network Wait, IDUBS in Cytestion tool	Outperformed TESTAR in fault detection and runtime
Manukonda (2024)	Inefficiency in Agile, limited browser/OS coverage	Selenium Grid (hub-node model)	Improved coverage and speed; complexity in managing distributed nodes
Inturi (2023)	Dynamic locators, test fragility, synchronization challenges	Selenium + TestNG, POM, structured reporting	Reduced regression time; environment instability affected results
Björkman (2023)	Synchronization, flaky tests, state reset issues	Playwright (auto-wait, getByRole), serial execution	Low learning curve, better stability; environment dependencies persist
Sezer (2023)	Oracle problem, high script maintenance, reproducibility	Quasi-experiment comparing Tosca vs. Testar	Tosca found more bugs but costly; Testar faster but less reliable
Jaganeshwari & Djodilatchoumy (2022)	Changing locators, large training data dependency	OpenCV + DNN, blob detection for widgets	98.4% accuracy, 35% RMSE reduction; dataset limitations
Lahtinen (2022)	Frequent UI changes, locator issues, third-party integrations	Selenium + xUnit, CI/CD, CSS selectors	CI/CD success in Azure DevOps; coverage limited by resources
Thooriqoh et al. (2021)	Script fragility, resource usage, weak reporting	SmartDriver (self-heal), headless browsers, AI/ML classifiers	Efficiency gains, improved flexibility; reporting relies on third-party tools
Nass et al. (2021)	Persistent automation challenges, ineffective tools	Systematic review of 49 studies	Most challenges remain unresolved; limited adoption of technical solutions

A. Conclusions

Based on the reviewed studies summarized in the table 1, several important conclusions can be drawn regarding GUI automation testing in dynamic web applications:

- Dynamic UI elements continue to pose the primary challenge in web testing, as most studies emphasize the difficulty of consistently identifying and interacting with elements that change due to AJAX calls, client side rendering, and real time DOM updates, which significantly compromises test reliability and maintainability.
- The Page Object Model (POM) combined with Selenium WebDriver is widely adopted by researchers to enhance test maintainability through improved modularity, yet this approach alone proves insufficient in addressing critical challenges such as synchronization issues and cross platform scalability limitations.
- Synchronization and timing issues represent recurring challenges in web testing, with test failures frequently occurring due to inadequate UI synchronization, prompting researchers to recommend solutions such as explicit/implicit waits, intelligent locators, and auto retry mechanisms like those found in Playwright to minimize flaky test occurrences.
- Artificial intelligence and machine learning are emerging as valuable enhancements in web testing, with researchers proposing intelligent locator strategies, image-based GUI detection, and machine learning algorithms for predictive maintenance to reduce manual effort and improve test robustness, although many of these advanced techniques remain in the experimental phase.
- Scriptless and low code testing tools like Cytestion and Scout are gaining traction by prototyping automated testing solutions that eliminate or reduce the need for traditional scripting, thereby lowering the technical barrier for testers and enhancing usability, though these approaches may encounter limitations when applied to complex enterprise environments with sophisticated testing requirements.
- The absence of standardization and persistent integration challenges continue to hinder web testing adoption, particularly in small teams and resource constrained projects where integrating with CI/CD pipelines and external services remains problematic, compounded by the lack of universal best practices for effectively testing dynamic content across different environments.
- Despite existing tools and patterns offering partial solutions to web testing challenges, there is a growing demand for more adaptive, context aware, and intelligent automation frameworks, with future research needing to prioritize the development of self-healing scripts, autonomous test generation capabilities, and advanced synchronization techniques to address the evolving complexities of modern web applications.

These conclusions underline the complexity of automating GUI testing for dynamic web applications and emphasize the necessity for continuous innovation in tooling, design methodologies, and intelligent automation techniques.

B. Recommendations

Based on the comparative analysis of the reviewed studies on GUI automation testing for dynamic web applications, the following recommendations are proposed to guide researchers, developers, and quality assurance teams:

- Implement robust test design patterns by leveraging the Page Object Model (POM) and keyword driven frameworks to significantly improve the maintainability and reusability of test scripts, particularly in dynamic environments where frequent UI element changes require adaptable and sustainable testing approaches.
- Implement intelligent element locators by utilizing self-healing locators, AI/ML driven element identification, and behavioral analysis techniques to enhance test robustness and minimize flakiness resulting from frequent UI updates and dynamic content changes.
- Leverage scriptless and visual testing tools such as Cytestion, Scout, or Testar to reduce technical barriers, automate exploratory testing

processes, and enable non-technical testers to effectively participate in GUI automation while maintaining comprehensive test coverage.

- Enhance synchronization mechanisms by implementing explicit and implicit waits, auto retry features, and network wait strategies to ensure proper coordination with dynamic content loading and minimize timing-related test failures in modern web applications.
- Integrate testing processes with CI/CD pipelines to enable continuous testing, which is crucial for maintaining test reliability and facilitating early defect detection in Agile and DevOps environments where rapid development cycles demand immediate feedback.
- Utilize distributed testing frameworks such as Selenium Grid or similar architectures to execute parallel tests across multiple platforms and browsers simultaneously, thereby enhancing test efficiency, scalability, and comprehensive coverage in diverse testing environments.
- Promote collaboration between developers and testers by fostering a DevTestOps culture that enhances communication, facilitates better test environment management, and addresses backend dependencies that impact UI test stability and overall software quality.
- Invest in comprehensive training and tool familiarization programs for QA teams to ensure proficiency with modern testing tools and automation frameworks, thereby reducing dependency on technical experts and minimizing maintenance overhead while improving overall testing efficiency.
- Address the limitations of current testing tools by conducting thorough evaluations based on scalability, integration ease, and dynamic UI handling capabilities, while advocating for the development of adaptive, AI-driven test generation systems that can better meet the evolving demands of modern web applications.

These recommendations aim to improve the reliability, efficiency, and adaptability of GUI automation testing processes in the face of ever-evolving web application technologies.

References

1. Björkman, M. (2023). Implementation of End-to-End testing in web application. *Umeå University, Sweden*, 1-62.
2. CAMARGO, L. S. (2024). A UTILIZAÇÃO DE TESTES AUTOMATIZADOS NO PROCESSO DE GARANTIA DE QUALIDADE DE INTERFACES DE APLICAÇÕES WEB. *INSTITUTO FEDERAL GOIANO - CAMPUS CERES*, 1-52.
3. Fadul, A. A., & Ibrahim, K. A. (2024). GUI Automation Testing Tools: a Comparative Study. *Excellence Journal for Engineering Sciences*, 44-62.
4. Fadul, A. A., & Ibrahim, K. A. (2025). Design and Implementation of a Framework for Automating Graphical User Interface (GUI) Testing Applied to Dynamic Web Applications. *Excellence Journal for Engineering Sciences*, 21-34.
5. Inturi, R. (2023). Implementing Test Automation with Selenium WebDriver. *West Virginia university*, 1-49.
6. Jayaraman, K. D. (2025). Automated Testing Frameworks: A Case Study Using Selenium and NUnit. *International Journal of Research Foundation of Hospital and Healthcare Administration* , 15-31.
7. K.JAGANESHWARI, & S.DJODILATCHOUMY. (2022). AN AUTOMATED TESTING TOOL BASED ON GRAPHICAL USER INTERFACE WITH EXPLORATORY BEHAVIOURAL ANALYSIS. *Journal of Theoretical and Applied Information Technology*, 6656-6666.
8. Lahtinen, M. (2022). Implementing UI regression testing process to existing web application. *Lappeenranta–Lahti University of Technology LUT*, 1-45.
9. Mahdy, A. A., & Atef Raslan. (2024). Towards applying An Automation Testing Framework for Educational Web Application. *Research Gate*, 1-7.
10. Manukonda, K. R. (2024). ENHANCING TEST AUTOMATION COVERAGE AND EFFICIENCY WITH SELENIUM GRID: A STUDY ON DISTRIBUTED TESTING IN

AGILE ENVIRONMENTS. *International Journal of Advanced Research in Engineering and Technology (IJARET)*, 119-127.

11. Mathew, S. (2024, 4 15). *manuscript*. Retrieved from preprints: https://www.preprints.org/manuscript/202404.0911/v1
12. Mothey, M. (2024). Test Automation Frameworks for Data-Driven Applications. *International Journal of Multidisciplinary Innovation and Research Methodology (IJMIRM)*, 361-381.
13. MOURA, T. S. (2024). AUTOMATIC SYSTEMATIC GUI TESTING FOR WEB APPLICATION. *UNIVERSIDADE FEDERAL DE CAMPINA GRANDE*, 1-159.
14. Nass, M. (2024). On overcoming challenges with GUI-based test automation. *Blekinge Institute of Technology*, 1-251.
15. Nass, M., Emil Alégroth, & Robert Feldt. (2021). Why many challenges with GUI test automation (will) remain. *Information and Software Technology*, 1-13.
16. Neelapu, M. (2025). Hybrid Testing Frameworks: Benefits and Challenges in Automation. *International Journal of Multidisciplinary Research and Growth Evaluation*, 2163-20168.
17. Sezer, B. (2023). An adaptation of an evaluation framework for software testing techniques and tools. *Utrecht*, 1-87.
18. Thooriqoh, H. A., Tiara Nur Annisa, & Umi Laili Yuhana. (2021). Selenium Framework for Web Automation Testing: A Systematic Literature Review. *JUTI: Jurnal Ilmiah Teknologi Informasi*, 65-76.