International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 5, Issue 6, 2026
DOI: https://doi.org/10.38124/ijsrmt.v5i6.1474

Model Context Protocol Approach to Reducing Test Development Effort in Web Automation

Oleksii Cherkashyn[1]

[1]IEEE Senior Member, Blynk Technologies Inc., Miami, Florida, USA

Publication Date: 2026/06/08

Abstract

The rapid evolution of web technologies and the increasing complexity of modern web applications have introduced significant challenges for test automation engineers. Frequent changes in user interfaces, business logic, and application architecture require continuous maintenance of automated test suites, leading to increased development effort and reduced productivity. At the same time, advances in Artificial Intelligence (AI) create new opportunities for improving software testing through intelligent automation and agent-based assistance. This study investigates the application of the Model Context Protocol (MCP) in web test automation, focusing on the WebdriverIO MCP Server as a mechanism for accelerating automated test development. The research analyzes the capabilities of MCP-enabled AI agents to interact with web applications and generate automated test scenarios in real-world environments. Furthermore, a two-machine development architecture is proposed, where the primary machine is used for main development activities and a secondary machine supports MCP-assisted test generation and exploratory automation, improving workflow efficiency and reducing interruptions. Experimental results demonstrate the practical applicability of the proposed approach. The findings indicate an average MCP session success rate of approximately 80%, suggesting that MCP-based automation can significantly reduce the effort required for developing automated web tests. The study highlights MCP as a promising framework for integrating AI into modern test automation workflows and provides recommendations for its adoption in software quality assurance processes.

Keywords: *WebdriverIO; MCP, Model Context Protocol; JavaScript; Web Automation.*

I. INTRODUCTION

Software testing plays a crucial role in the software development lifecycle, ensuring that end users receive more reliable applications and services. Test automation is a branch of software quality assurance that differs from manual testing in that test execution is performed by software tools rather than human testers (Moń & Pańczyk, 2025). Ensuring application quality through efficient testing has become increasingly critical in today's fast-paced software development environment. As organizations strive to accelerate product releases, the limitations of traditional manual testing methods become more apparent. While manual testing was once sufficient to ensure software quality, it is no longer adequate given the rapid evolution of software systems, growing complexity, and the demand for faster delivery cycles. Test automation systems have therefore been widely adopted. However, even these advanced tools face challenges in managing the increasing complexity and scale of modern applications. The integration of artificial intelligence (AI) into test automation is transforming the field by improving scalability, accuracy, and overall efficiency (Sangaraju, 2024). The growing adoption of Artificial Intelligence (AI) in software testing has led to the emergence of a new class of AI-driven testing tools that utilize machine learning (ML), natural language processing (NLP), and computer vision to automate essential testing activities. These include AI-based test generation, self-healing test automation, visual regression testing, and automated failure analysis (Garousi et al., 2024). Additionally, test case generation remains a crucial but challenging aspect of software development (Alagarsamy et al., 2024). High-quality test cases and test suites are essential for achieving reliable testing results and increasing confidence in software quality (Tran et al., 2025). Certainly, AI can generate high-quality test cases; however, it does not fully solve the problem of independently creating automated test scripts. In this context, a new approach such as the MCP server can provide additional support and bridge this gap.

The Model Context Protocol (MCP) is an emerging interoperability standard designed to bridge

Cherkashyn, O. (2026). Model Context Protocol Approach to Reducing Test Development Effort in Web Automation. *International Journal of Scientific Research and Modern Technology*, *5*(6), 1–6. https://doi.org/10.38124/ijsrmt.v5i6.1474

the gap between autonomous, reasoning-driven AI systems and the deterministic, schema-bound infrastructure of enterprise software (Venkiteela, 2025).

In this study, an analysis is conducted on the use of the WebdriverIO MCP server for writing automated web tests. WebdriverIO is a modern test automation framework for web and mobile applications that provides a simple and flexible API for test development and is considered an efficient alternative in terms of both performance and usability (Cherkashyn, 2025). Node.js applications commonly rely on a large number of publicly available third-party packages distributed through the Node Package Manager (npm) registry, which is a vast repository offering over one million reusable libraries (Sun et al., 2021). Among these libraries is WebdriverIO, along with its associated dependencies. In this study, the WebdriverIO MCP server is utilized in conjunction with widely adopted development tools, including the Visual Studio Code IDE, a lightweight and extensible development environment with rich built-in functionality and an extensive extension ecosystem (Tan et al., 2024), and LLM-powered AI assistants such as GitHub Copilot (Sergeyuk et al., 2025), to support and enhance the automation testing workflow.

The key contribution of this research is twofold. First, it expands the current literature on the application of the Model Context Protocol (MCP) in web test automation, an area that remains largely underexplored due to the novelty of the technology. Second, it provides a systematic analysis of the WebdriverIO MCP Server as an AI-assisted framework for automated web test development. The study evaluates its effectiveness in supporting test creation activities and reducing test development effort, thereby contributing both theoretical and practical insights into the adoption of MCP-based approaches in modern software testing environments.

II. LITERATURE REVIEW

In the broader context of artificial intelligence and software engineering, the Model Context Protocol (MCP) represents a foundational component of AI-native application architectures by improving interoperability and mitigating ecosystem fragmentation. It was initially introduced in late November 2024 (Hasan et al., 2026). Due to the recent introduction of the Model Context Protocol (MCP), its adoption and implications within the field of web test automation remain insufficiently addressed in current scientific research.

According to the study of Wang et al. (2026), the authors propose an MCP-enabled framework called PTFusion that enhances automated web penetration testing through coordinated interaction between multiple LLM-driven agents and specialized security tools. The proposed approach uses the Model Context Protocol (MCP) as a unified mechanism for invoking heterogeneous penetration-testing tools, enabling more efficient integration of reconnaissance and exploitation processes within a single workflow. Furthermore, the framework introduces a semi-decentralized multi-agent architecture in which different agents are responsible for strategic planning and tactical execution, improving decision-making during complex testing scenarios. To address the problem of fragmented and noisy outputs generated by security tools, the authors implement a context-aware knowledge fusion mechanism based on a dynamic knowledge graph and preference-based chain-of-thought reasoning. Experimental results demonstrate that the proposed MCP-based architecture achieves higher stability, more accurate command execution, and better task completion rates compared with existing LLM-assisted penetration-testing approaches.

According to the study of Saleh et al. (2026), the authors propose MCP-Web-Curl, a Model Context Protocol (MCP) server designed to provide large language model–based agents with structured access to web resources and REST APIs. The proposed architecture enables AI-driven systems to perform browser-based web scraping, API inspection, documentation retrieval, and intelligent search operations through a unified MCP interface. In the context of web testing and automated software engineering tasks, the framework allows agents to dynamically obtain external information, analyze web resources, and interact with online services that are not available within the model's internal knowledge base. Furthermore, the authors introduce mechanisms for token-efficient resource management, including content truncation control and request filtering, which improve the predictability and reliability of agent execution. As a result, MCP-Web-Curl is presented as a reusable infrastructure layer that enhances the capabilities of autonomous testing and agentic coding environments by extending LLMs with controlled web and API interaction capabilities.

According to the study of Binder et al. (2026), the authors propose an MCP-based architecture for integrating large language model agents with web testing environments through structured tool orchestration mechanisms. The work emphasizes the idea of representing web testing operations as executable, context-aware tool calls exposed via an MCP server, which allows AI agents to interact with web applications in a more standardized and controllable way. Instead of relying on fragile DOM-based interaction or isolated automation scripts, the proposed approach introduces a unified MCP interface that enables dynamic discovery and execution of testing actions as modular services. In this framework, test generation, execution, and result interpretation can be coordinated through MCP-enabled toolchains, improving reproducibility and automation consistency in web testing pipelines. Overall, the authors position MCP servers as an intermediate abstraction layer that bridges LLM-driven agents and real web testing infrastructures, enhancing scalability and controllability of automated testing workflows.

Luo et al. (2025) propose MCP-Universe as a real-world evaluation framework for LLM agents interacting with Model Context Protocol (MCP) servers, where web

automation is treated as one of the core tool-usage environments. In the context of web automation, they incorporate dedicated MCP servers such as browser automation and web searching services, enabling agents to execute multi-step interactions with real web interfaces rather than simulated tasks. The authors frame web automation as a long-horizon tool-use problem, where agents must continuously interact with browser-based MCP tools (e.g., navigation, page actions, information extraction) under evolving context constraints. They emphasize that these interactions are significantly more complex than static web testing because each step changes the environment and increases reasoning load. Their results show that current LLM agents struggle in these MCP-driven web automation scenarios, primarily due to context growth across interaction steps and unfamiliarity with tool interfaces, which leads to frequent execution errors and degraded task completion.

While the existing body of work demonstrates significant progress in applying the Model Context Protocol (MCP) to AI-driven tool orchestration, web access, and automated testing environments (Hasan et al., 2026; Wang et al., 2026; Saleh et al., 2026; Binder et al., 2026; Luo et al., 2025), these studies primarily focus on either architectural design of MCP-enabled systems or benchmark-level evaluations of agent performance in generic web automation scenarios. In contrast, they do not specifically investigate MCP integration within real-world industrial web testing toolchains or its impact on day-to-day test automation workflows used by QA engineers. The present study addresses this gap by focusing on the practical application of an MCP-based approach for reducing effort in web test automation through the direct use of a WebdriverIO MCP server integrated into the Visual Studio Code IDE environment with GitHub Copilot assistance. Unlike prior research that evaluates MCP in abstract agent-based settings or security-oriented frameworks, this work empirically analyzes MCP usage in a developer-centric workflow, comparing manual test script creation with MCP-assisted test generation. Furthermore, it examines the usability, efficiency gains, and limitations of MCP adoption in real testing scenarios, providing a more practice-oriented perspective on its effectiveness in software quality assurance pipelines.

III. METHODOLOGY

➢ *Experimental Environment*

In this study, two local machines were utilized, running Windows 10 Pro x64 and macOS Sequoia, respectively. The test execution was performed using Node.js version v22.22.3. Google Chrome browser version 148.0.7778.217 served as the primary execution environment. The automation framework was implemented using WebdriverIO version 8, with JavaScript as the primary programming language. The following JSON structure presents the core configuration of libraries and project dependencies used in the experimental setup.

```
{
"name": "app_name",
"type": "module",
"devDependencies": {
"@wdio/cli": "^8.32.3",
"@wdio/local-runner": "^8.32.3",
"@wdio/mocha-framework": "^8.16.3",
"@wdio/spec-reporter": "^8.16.3",
"chromedriver": "^149.0.1",
"@wdio/mcp": "^3.4.0",
"wdio-chromedriver-service": "^8.1.1"
},
"scripts": {
"test": "wdio run ./wdio.conf.js"
},
}
```

Additionally, the study incorporates the WebDriverIO MCP Server (Model Context Protocol) version 3.4.0. This component is used to enable seamless integration between the test automation framework and AI-assisted development tools. The following JSON configuration defines the MCP setup used in the Visual Studio Code environment.

```
{
"servers": {
"wdio-mcp": {
"command": "npx",
"args": [
"-y",
"@wdio/mcp@3.4.0"
]
}
},
"inputs": [],
}
```

➢ *Proposed Structure for Working with the MCP Server*

Modern software development is characterized by the widespread adoption of agile methodologies, continuous integration, and continuous deployment practices. These trends require testing activities to keep pace with accelerated release cycles and deliver timely feedback regarding software quality, reliability, and stability.

In rapidly evolving web applications, where changes are introduced frequently, a substantial portion of test automation effort is often devoted to maintaining existing test suites rather than developing new ones. Test engineers routinely spend significant time on test refactoring, CI/CD pipeline maintenance, debugging failed executions, and manually reproducing failed test scenarios to determine whether failures are caused by software defects, test instability, regressions, or intended changes in application behavior. As a result, the resources available for expanding automated test coverage are often limited.

To address this challenge, the proposed approach utilizes two separate local machines. The primary machine is dedicated to the maintenance and evolution of the

existing automation framework, including test refactoring, debugging activities, and CI/CD-related tasks. The secondary machine is exclusively used for the development of new automated tests through the MCP server.

The use of a dedicated secondary machine provides several practical advantages. In many real-world web applications, the feasibility of performing multiple parallel activities on a single workstation is constrained by the underlying client–server architecture, session management mechanisms, authentication workflows, cookies, and other platform-specific factors. Consequently, executing maintenance activities and developing new test cases simultaneously on the same web platform, even with a multi-monitor setup, may be technically challenging or inefficient. By separating these activities across two independent environments, it becomes possible to reduce workflow interruptions and improve the overall productivity of test automation development.

The proposed two-machine architecture is presented in Figure below.

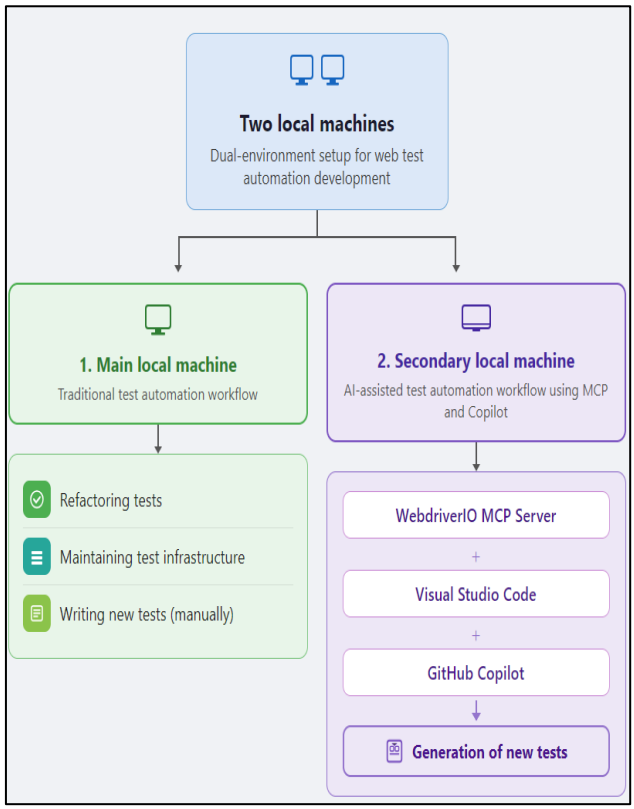

Fig 1 Proposed Approach for Improving the Performance of Web Test Automation Development

IV. RESULTS AND DISCUSSION

As mentioned in the article, the study utilized the WDIO MCP Server version 3.4.0, which was selected as the most widely adopted release. According to the package page on the npm registry, this version has recorded approximately 1,275 downloads. The analysis further demonstrated that the MCP server exposes a total of 29 tools, namely: click_element, close_session, delete_cookies, drag_and_drop, emulate_device, execute_script, get_accessibility_tree, get_app_state, get_contexts, get_cookies, get_elements, get_screenshot, get_tabs, hide_keyboard, launch_chrome, list_apps, navigate, rotate_device, scroll, set_cookie, set_geolocation, set_value, start_session, swipe, switch_context, switch_frame, switch_tab, tap_element, and upload_app. The study also identified the most frequently used tools as click_element, close_session, launch_chrome, navigate, set_value, and start_session in the context of web automation.

The experiments were conducted on three React-based web applications, each requiring user authentication via a login account.

The results showed that the average time required to formulate a prompt in the VS Code AI chat to execute a login session was approximately 60 seconds. This workflow typically included starting a session, entering credentials into email and password fields, clicking the login button, and performing a set of post-login UI validations to verify successful authentication.

Furthermore, the study demonstrated that such prompts, when structured as detailed execution instructions, can be stored as separate files and later reused by referencing them in the chat via a file-loading command. For example, a prompt such as “open Chrome browser and execute these steps via MCP WDIO, instructions located in run.txt file” can be used to trigger predefined workflows.

However, in practice, this approach demonstrated an average success rate of approximately 50%, as the AI agent may intermittently request clarification or re-confirmation of individual execution steps during runtime. The study was conducted over 30 consecutive sessions, with each experiment repeated three times to improve result reliability. Following each session, the generated test scenarios were manually executed to validate the correctness of the AI-generated actions and outcomes.

The findings indicate an average MCP session success rate of approximately 80%. However, this rate is strongly influenced by the quality, clarity, and level of detail provided in the prompt. In particular, including portions of element locators within the prompt significantly improved execution accuracy. Elements containing unique and clearly identifiable text were generally located and interacted with successfully. In contrast, interactions involving icons, graphical controls, or elements without visible text proved more challenging for the AI agent. In such cases, providing either a complete or partial locator was often necessary to achieve reliable execution. Dropdown fields also emerged as a notable challenge. While the AI agent was typically able to open a dropdown component, successful selection of the desired option frequently required explicitly specifying the locator of the target item within the prompt. These observations suggest that the effectiveness of MCP-based web automation can be substantially improved through the

inclusion of precise element identification information, particularly for non-textual UI components and dynamic interface elements.

The time required to create high-quality and comprehensive test prompts remains significant. In this context, productivity can be substantially improved through the use of speech-to-text functionality provided by VS Code extensions, allowing testers to dictate detailed instructions rather than manually typing them.

Another important limitation observed during the study is that the generated tests are typically created without adherence to the Page Object Model (POM) architecture. Consequently, when parts of the application are already covered by an existing automation framework, additional interaction with the AI agent is required to refactor the generated code and align it with established project structures and coding standards.

Furthermore, a notable limitation of the MCP server is its inability to effectively refactor or maintain existing automated tests. Its capabilities are primarily focused on exploratory interaction with the application under test and the generation of new test cases. As a result, activities such as updating existing test suites, improving test architecture, eliminating code duplication, and performing large-scale refactoring still require manual effort or the use of additional AI-assisted development workflows outside the MCP execution environment.

V. CONCLUSION

This study investigated the applicability of the WebdriverIO MCP Server as an AI-assisted approach for reducing test development effort in web automation. The experimental results demonstrated that MCP-based workflows can effectively support the creation of new automated web tests through natural-language interaction with AI agents. Across the conducted experiments, the average MCP session success rate reached approximately 80%, indicating the practical viability of the approach.

The findings suggest that the effectiveness of MCP-driven automation is highly dependent on prompt quality and the availability of precise element identification information. While elements containing unique textual content were generally handled successfully, interactions involving icons, dynamic components, and dropdown menus often required additional locator guidance.

The study also identified several limitations. Although the MCP server proved effective for exploratory application interaction and the generation of new test scenarios, its capabilities for maintaining and refactoring existing test suites remain limited. In addition, the generated tests do not inherently follow architectural patterns such as the Page Object Model, requiring further refinement by automation engineers.

Overall, the results indicate that MCP technology can serve as a valuable complement to conventional test automation frameworks by accelerating the development of new automated tests and reducing manual effort during the initial stages of test creation. As MCP is still an emerging technology, further research is required to improve execution reliability, enhance support for established test automation architectures, and evaluate its applicability in large-scale industrial software testing environments.

REFERENCES

[1]. Venkiteela, P. (2025). The New Interoperability Paradigm: Model Context Protocol (MCP), APIs, and the Future of Agentic AI. Computer Fraud and Security. https://doi.org/10.52710/cfs.817.

[2]. Moń, M., & Pańczyk, B. (2025). A comparative analysis of web application test automation tools. Journal of Computer Sciences Institute, 35, 159–165. https://doi.org/10.35784/jcsi.7119.

[3]. Sangaraju, V. V. (2024). AI-augmented test automation: Leveraging Selenium, Cucumber, and Cypress for scalable testing. EPH-International Journal of Science and Engineering, 10(4). https://doi.org/10.53555/ephijse.v7i2.278.

[4]. Garousi, V., Joy, N., Jafarov, Z., Keleş, A. B., Değirmenci, S., Özdemir, E., & Zarringhalami, R. (2024). AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations. arXiv preprint. https://doi.org/10.48550/arXiv.2409.00411.

[5]. Alagarsamy, S., Tantithamthavorn, C., Aleti, A. (2024). A3Test: Assertion-Augmented Automated Test case generation. Information and Software Technology, 176, 107565. https://doi.org/10.1016/j.infsof.2024.107565.

[6]. Tran, H. K. V., Ali, N. bin, Unterkalmsteiner, M., Börstler, J., & Chatzipetrou, P. (2025). Quality attributes of test cases and test suites – importance & challenges from practitioners' perspectives. Software Quality Journal, 33, Article 9. https://doi.org/10.1007/s11219-024-09698-w.

[7]. Cherkashyn, O. (2025). Application Test Automation in Headless Android Emulator. In Proceedings of the International Conference on Applied Innovations in IT (ICAIIT). http://dx.doi.org/10.25673/123064.

[8]. Sun, H., Rosà, A., Bonetta, D., & Binder, W. (2021). Automatically assessing and extending code coverage for NPM packages. In 2021 IEEE/ACM International Conference on Automation of Software Test (AST 2021), 40–49. https://doi.org/10.1109/AST52587.2021.00013.

[9]. Tan, J., & Chen, Y., & Jiao, S. (2024), Visual Studio Code in Introductory Computer Science Course: An Experience Report Paper presented at 2024 ASEE Annual Conference & Exposition, Portland, Oregon. https://doi.org/10.18260/1-2--48259.

[10]. Sergeyuk, A., Golubev, Y., Bryksin, T., & Ahmed, I. (2025). Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. Information and Software Technology,

178, 107610. https://doi.org/10.1016/j.infsof.2024.107610

[11]. Hasan, M. M. H., Li, H., Fallahzadeh, E., Rajbahadur, G. K., Adams, B., & Hassan, A. E. (2026). Model Context Protocol (MCP) at first glance: Studying the security and maintainability of MCP servers. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3814959.

[12]. Wang, W., Gu, H., Wu, Z., Chen, H., Chen, X., & Shi, F. (2026). PTFusion: LLM-driven context-aware knowledge fusion for web penetration testing. Information Fusion, 127(Part A), Article 103731. https://doi.org/10.1016/j.inffus.2025.103731

[13]. Saleh, R. Z., & Lubis, M. (2026). Design and Implementation of MCP-Web-Curl: A Model Context Protocol Server for Web and API Access in Agentic Coding Assistants. Jurnal Teknik Informatika, 19(1), 122–134. https://doi.org/10.15408/jti.v19i1.49625.

[14]. Binder, E., Li, X., & Janes, A. (2026). An Agent-Based Approach to Automating Software Performance Testing. ICPE Companion '26: Companion of the 17th ACM/SPEC International Conference on Performance Engineering, pp. 55–61. https://doi.org/10.1145/3777911.3801107.

[15]. Luo, Z., Shen, Z., Yang, W., Zhao, Z., Jwalapuram, P., Saha, A., Sahoo, D., Savarese, S., Xiong, C., & Li, J. (2025). MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers. arXiv preprint. https://arxiv.org/abs/2508.14704.