Enhancing Web Test Automation Through Screenshot and Video-Based Execution Reporting

Oleksii Cherkashyn
Blynk Technologies Inc., Brickell Avenue 951, 33131 Miami, FL, USA
oleksii.sciencepapers@gmail.com

Keywords: Video Reporter, Screenshot Reporter, Web Application Testing, Web Test Automation, WebdriverIO, JavaScript.

Abstract: Automated web testing increasingly relies on comprehensive reporting mechanisms to facilitate bug reproduction and analysis. This study focuses on enhancing test automation through screenshot and video-based execution reporting using the WebdriverIO framework. The built-in method saveScreenshot was employed for capturing screenshots at each step of the test execution, while the Video Reporter library was used to record videos for failed tests. The study demonstrates that both GUI and headless modes of the Chrome browser do not affect the quality of screenshots and videos. Captured screenshots have an average size of approximately 61.5 KB to 136.9 KB, and videos range from 18.4 KB to 71.7 KB, making them suitable for execution on low-budget or resource-limited remote servers. Furthermore, with Jenkins supporting PNG screenshots and WEBM video playback, visual analysis can be performed directly within the CI/CD interface. The results confirm that the visual quality of both screenshots and video recordings is sufficient to identify the exact point of test failure and to reproduce the corresponding bugs, providing practical guidance for improving test reporting and debugging in automated web testing environments.

1 INTRODUCTION

Web applications play a crucial role in modern companies and organizations. They enhance user engagement by providing simple and accessible ways for interaction, allowing businesses to communicate effectively with their users. Given their importance, thorough and continuous testing is essential to ensure high quality. However, this process introduces several challenges. As a result, testing web applications is generally more complex than testing other types of software, requiring efficient tools and well-designed methodologies [1].

Manual testing is generally considered less reliable compared to automated testing carried out with specialized tools and scripts. In addition, it tends to require more time and effort, whereas automated approaches enable faster execution of test cases. As a result, automation significantly improves testing speed and efficiency. Its implementation allows developers to detect defects more quickly and streamline the overall testing process. Moreover, automated testing supports effective regression testing by identifying issues that may arise after changes are introduced into the codebase [2].

A critically important aspect of automated web testing is the reporting of test results, particularly in real time. Real-time insights enable teams to identify patterns, detect anomalies, and highlight areas that require improvement [3]. For additional analysis of failed tests, screenshot and video reporting tools can be used, as they provide a faster and more intuitive way to identify the location and root cause of errors.

In the context of this study, the WebdriverIO framework [4] is utilized as a robust and reliable solution for automated testing. WebdriverIO, a Node.js-based framework available via npm [5], provides flexibility and seamless compatibility with a wide variety of testing tools and environments. The official version of the WebdriverIO video reporter (wdio-video-reporter) [6] is used in the test framework to record videos of test executions. In addition, standard WebdriverIO methods are applied to capture screenshots on test failures, ensuring that both video and image artifacts are available for debugging and reporting purposes. This approach allows detailed visualization of test results and facilitates faster identification of issues.

This paper contributes to the practices of test analysis and debugging by utilizing screenshots and

video reporting for failed tests, implemented using the WebdriverIO library.

The research question addressed in this study is whether screenshot and video-based execution reporting can improve the efficiency of failure analysis and bug reproduction in automated web testing while remaining suitable for execution in resource-constrained CI/CD environments. In particular, the study investigates the feasibility of integrating lightweight visual reporting mechanisms into automated testing pipelines without significantly increasing storage or computational requirements.

The novelty of this work lies not in the introduction of a new reporting tool, but in the experimental evaluation and practical integration of screenshot and video-based execution reporting within the WebdriverIO framework under different browser execution modes. Unlike existing studies that primarily describe reporting mechanisms conceptually, this paper provides a quantitative analysis of artifact sizes, evaluates compatibility with Jenkins CI/CD environments, and demonstrates that lightweight visual artifacts generated in both GUI and headless browser modes remain sufficient for effective debugging and failure reproduction. Therefore, the study contributes practical evidence regarding the applicability of visual execution reporting for low-budget and resource-limited automated testing infrastructures.

2 RELEVANCE

Reproducing bugs depends on the information contained in bug reports to replicate the issues described. Functionally, a bug report generally consists of key elements such as Steps to Reproduce (S2R), Expected Behavior (EB), and Observed Behavior (OB). Regarding media types, reports may include textual descriptions, images, and videos or GIFs. Images refer specifically to UI screenshots or any static visual content provided by users that can aid the bug reproduction process [7].

In their study, Mallipeddi, Yaqoob, Khan, Mehmood, Mylonas, and Pitropakis describe how testers manually follow test plans, execute test cases, and capture evidence, such as screenshots, comparing actual results with expected outcomes to identify discrepancies and unexpected behavior [8].

Based on the analysis, the following conclusions can be drawn:

- Screenshots and videos of failed tests assist in manually reproducing bugs;
- They provide a visual understanding of differences between previous and current builds;
- Unlike stack traces or textual descriptions of potential errors, visual evidence offers a faster and more intuitive way to locate bugs, thereby saving time.

Thus, the topic of reproducing failed automated tests is relevant, highlighting the need to study and improve existing practices related to screenshot and video reporters.

3 ANALYSIS OF CURRENT RESEARCH

Overall, there is a lack of scientific research focused on the use of screenshot or video reporters for failed automated tests specifically with the WebdriverIO library. Therefore, studies related to other frameworks and existing practices were examined.

In their study, Mon and Panczyk note that Cypress stands out for its simplicity and speed, as well as its built-in screenshot and video capture capabilities, which simplify the testing process. Playwright, on the other hand, is a modern tool offering advanced features, stability, and multi-browser support. Selenium supports the largest number of programming languages but lacks native screenshot and video capture functionality. The authors also provide a table indicating that both Cypress and Playwright support screenshot and video reporters. However, their study does not include an analysis of how these reporters are actually used in practice [9].

In their research, Zhao, Talebipour, Baral, Park, Yee, Khan, Brun, Medvidovic, and Moran introduce *Avgust*, an automated approach that uses video recordings of app executions to generate usage-based UI test cases. They propose processing these videos with neural image-understanding models to automatically infer user interaction sequences from visual state changes captured in screenshots and frames. This enables the synthesis of an app-agnostic model of user behavior, which can then be used to generate new test cases for unseen applications. Unlike traditional test automation that focuses on code coverage or crash detection, their method leverages visual artifacts from videos as a primary source for understanding real usage patterns. The authors show that a significant portion of the tests Avgust generates successfully exercise desired behaviors, demonstrating the effectiveness of using visual evidence (screenshots/video) to construct meaningful tests. Overall, their work highlights how

visual reporters of UI interactions can be used not just for evidence, but as input data to drive automated test creation [10].

Wang, Li, Wang, Menzies, and Wang in their study propose leveraging screenshots in bug reports to improve duplicate bug report detection. They suggest extracting and analyzing visual features from screenshots and combining them with textual information to better identify similar issues. Their approach demonstrates that visual data provides complementary context that is often missing in text-only descriptions, leading to improved detection accuracy. From a web automation perspective, this implies that screenshots captured during test execution can be utilized not only for reporting but also for automated failure analysis. Overall, their work highlights the potential of integrating screenshot reporters with intelligent techniques to enhance bug triaging and reduce redundancy [11].

Ashrafi, Bouktif, and Mediani in their study propose enhancing bug reports by leveraging screenshots as a primary source of information. They suggest automatically extracting meaningful visual features from screenshots to better represent the state of the user interface at the moment of failure. Their approach aims to complement or clarify textual descriptions, reducing ambiguity and improving bug understanding. From a web automation perspective, this implies that screenshots generated during test execution can be used not only for reporting but also for automated analysis and interpretation of failures. Overall, their work highlights the potential of transforming screenshot reporters into intelligent tools for improving bug reproducibility and diagnosis [12].

Existing research and practices primarily focus on using screenshots and video reporters from failed tests as visual tools for reproducing potential bugs, debugging, and analyzing issues or changes in the platform after development. This study extends and complements these practices by applying them specifically within the WebdriverIO framework.

Unlike prior research that mainly investigates screenshots and videos as auxiliary debugging evidence or as datasets for intelligent processing techniques, this study introduces a resource-oriented evaluation of visual execution reporting within WebdriverIO-based automated testing. The scientific contribution of the work lies in demonstrating that lightweight screenshot and video artifacts generated during automated test execution maintain sufficient diagnostic quality for failure analysis and bug reproduction, even in headless execution environments and resource-constrained CI/CD systems. Additionally, the study provides quantitative evidence regarding artifact sizes and CI/CD compatibility, which is largely absent in existing literature. This shifts the focus from conceptual usage of visual reporting toward experimentally validated applicability in practical industrial automation infrastructures.

4 SOFTWARE CONFIGURATION

4.1 Test Infrastructure Configuration

The study was conducted using Node.js version 20.19.4 in combination with WebdriverIO version 8 for automated testing. Mocha was employed as the test runner, with test scripts written in JavaScript [13]. Mocha offers a flexible and adaptable framework suitable for a variety of testing strategies, including support for asynchronous testing, which is crucial for modern web applications that rely on dynamic content [13]. Chromedriver is used to control the Chrome browser during automated testing [14]. It serves as the WebDriver implementation that translates test commands into browser actions, enabling the execution of test scripts and interaction with web elements. Using chromedriver ensures that the tests run reliably on the Chrome browser and allows integration with the WebdriverIO framework for end-to-end automation. A complete list of dependencies and libraries is provided in the package.json file, included in Appendix (Listing A1).

4.2 Screenshot and Video Reporter Configuration

In general, the key settings for screenshot and video reporters are defined in the framework's configuration file, wdio.conf.js. For screenshots, the built-in framework method saveScreenshot is sufficient. To facilitate storage and later analysis, each screenshot can be named using the exact date and time. This process can be assisted by libraries such as Moment.js [15]. The main configuration of the reporters in the framework is provided in Appendix (Listing A2). In the video reporter, the videoSlowdownMultiplier option was set to 10, which analysis showed to be a sufficient value. This setting is used to adjust the playback speed of the recorded videos - higher values produce slower videos, while lower values produce faster videos. Acceptable values range from 1 to 100.

5 RESULTS AND DISCUSSION

In this study, the latest stable version of the Chrome browser, Version 145.0.7632.117 (Official Build) (64-bit), was used, along with the latest chromedriver 146.0.0. Automated tests were executed on a local Windows 10 machine as well as on a remote Ubuntu server. The tests were conducted on three different React web applications and organized into three separate test packs containing 200, 100, and 50 tests, respectively. The browser was configured with a resolution of 1920 × 1080 and launched with the start maximized option.

The tests were executed in both GUI and headless modes of the browser. The study showed that the screenshot and video reporters do not affect the mode, producing outputs of the same quality and resolution in both cases.

Regarding screenshots taken during the tests, in the case of a failed test, the screenshot accurately captured the exact point at which the test failed. The screenshots are of high quality, and when comparing 100 screenshots, their file sizes ranged from 61.5 KB - 136.9 KB.

Regarding the video reporter, as indicated in Appendix 2, video recording is performed only for failed tests. However, screenshots are captured at every stage of each test regardless of its outcome. For each individual test, a folder is created with the name of the test, and all corresponding screenshots are saved. The number of screenshots depends on the size of the test, but on average, 5 to 30 screenshots are generated. The file sizes of these screenshots range from approximately 31.0 KB to 154.3 KB.

In the case of a failed test, these screenshots are compiled into a single WEBM video file, with sizes ranging from approximately 18.4 KB to 71.7 KB. The video quality is moderate but sufficient to understand the test execution flow and the conditions under which the test fails. Tests were also executed via Jenkins on an Ubuntu server, and analysis showed that both screenshots and videos can be viewed directly through the Jenkins UI.

The test packs were executed 10 times as part of this study. The results also indicate that a key limitation is that the framework does not automatically delete all screenshots captured by the video reporter during test execution. Thousands of screenshots can be generated in a single day, so it is necessary to include scripts to remove these files.

Although the study primarily focuses on the practical applicability of screenshot and video-based reporting, the experimental evaluation demonstrates several measurable characteristics of the proposed approach. In particular, the generated artifacts maintained stable quality across different execution environments, including GUI and headless browser modes, as well as Windows and Ubuntu operating systems. Furthermore, the relatively small file sizes of screenshots and WEBM videos indicate that the approach can be applied in CI/CD infrastructures with limited storage and computational resources without introducing significant overhead.

From a debugging perspective, the collected screenshots and videos enabled rapid identification of the exact failure step and the sequence of actions preceding the error. While the present work does not include a formal user study or quantitative measurements of debugging speed, repeated execution of the test packs demonstrated that visual reporting artifacts consistently improved the interpretability of failed test executions compared to log-only reporting approaches. Therefore, the proposed method can reduce the effort required for reproducing defects and analyzing unstable automated tests in practical testing environments.

A potential threat to validity is that the experiments were conducted using a specific configuration involving the Chrome browser, WebdriverIO framework, React-based applications, and Jenkins CI/CD infrastructure. Different browser engines, operating systems, or reporting plugins may produce different storage characteristics or execution overhead. However, because the proposed approach relies on standard WebdriverIO APIs and widely supported PNG and WEBM formats, the findings are expected to be transferable to similar modern web automation environments.

6 CONCLUSIONS

Although there are studies on screenshot and video reporters in automated web testing, this research complements existing practices in the context of using the WebdriverIO library. Based on the results of this study, several conclusions can be drawn:

- The GUI and headless modes of the Chrome browser do not affect the quality of the screenshots and videos captured for failed tests;
- The saved screenshots have an average size of approximately 61.5 KB to 136.9 KB, while the videos range from 18.4 KB to 71.7 KB, indicating relatively small file sizes. This makes them suitable for running tests on low-budget or resource-limited remote servers;
- Since Jenkins supports viewing PNG screenshots and WEBM video files, visual analysis can be performed directly within the Jenkins interface;

- The visual quality of the screenshots and video recordings is sufficient to clearly identify the point at which a test failed and to reproduce the bug.

REFERENCES

[1] S. Balsam and D. Mishra, "Web application testing - Challenges and opportunities," Journal of Systems and Software, vol. 219, Jan. 2025, art. 112186, [Online]. Available: https://doi.org/10.1016/j.jss.2024.112186.

[2] H. At Thooriqoh, T. N. Annisa and U. L. Yuhana, "Selenium Framework for Web Automation Testing: A Systematic Literature Review," JUTI: Jurnal Ilmiah Teknologi Informasi, vol. 19, no. 2, Jul. 2021, [Online]. Available: https://doi.org/10.12962/j24068535.v19i2.a1021.

[3] R. M. Salagundi and S. R. Savitha, "Automated reporting and statistical analysis of test case results in continuous integration: A custom dashboard approach," World Journal of Advanced Engineering Technology and Sciences, vol. 13, no. 1, pp. 027-033, 2024, [Online]. Available: https://doi.org/10.30574/wjaets.2024.13.1.0374.

[4] O. Cherkashyn, "Application Test Automation in Headless Android Emulator," Proceedings of International Conference on Applied Innovation in IT, vol. 13, no. 5, pp. 445-452, 2025, [Online]. Available: https://doi.org/10.25673/123064.

[5] H. Sun, A. Rosà, D. Bonetta and W. Binder, "Automatically Assessing and Extending Code Coverage for NPM Packages," in 2021 IEEE/ACM International Conference on Automation of Software Test (AST 2021), pp. 40-49, 2021, [Online]. Available: https://doi.org/10.1109/AST52587.2021.00013.

[6] "Video Reporter," [Online]. Available: https://webdriver.io/docs/wdio-video-reporter/.

[7] D. Wang, Z. Zhang, S. Feng, W. G. J. Halfond and T. Yu, "An Empirical Study on Leveraging Images in Automated Bug Report Reproduction," in Proceedings of the 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), pp. 123-133, Apr. 2025, [Online]. Available: https://doi.org/10.1109/MSR66628.2025.00019.

[8] S. Mallipeddi, M. Yaqoob, J. A. Khan, T. Mehmood, A. Mylonas and N. Pitropakis, "AutoQALLMs: Automating Web Application Testing Using Large Language Models (LLMs) and Selenium," Computers, vol. 14, no. 11, p. 501, Nov. 2025, [Online]. Available: https://doi.org/10.3390/computers14110501.

[9] M. Mon and B. Panczyk, "A Comparative Analysis of Web Application Test Automation Tools," Journal of Computer Sciences Institute, vol. 35, pp. 159-165, Jun. 2025, [Online]. Available: https://doi.org/10.35784/jcsi.7119.

[10] Y. Zhao, S. Talebipour, K. Baral, H. Park, L. Yee, S. A. Khan, Y. Brun, N. Medvidovic and K. Moran, "Avgust: Automating Usage Based Test Generation from Videos of App Executions," in Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022), pp. 421-433, Nov. 2022, [Online]. Available: https://doi.org/10.1145/3540250.3549134.

[11] J. Wang, M. Li, S. Wang, T. Menzies and Q. Wang, "Images don't lie: Duplicate crowdtesting reports detection with screenshot information," Information and Software Technology, vol. 110, pp. 139-155, Jun. 2019, [Online]. Available: https://doi.org/10.1016/j.infsof.2019.03.003.

[12] A. Ashrafi, M. Bouktif and R. Mediani, "ImageR: Enhancing Bug Report Clarity by Screenshots," arXiv preprint arXiv:2505.01925, 2025, [Online]. Available: https://arxiv.org/abs/2505.01925.

[13] E. Kesavan, "Comparative Study of JavaScript-Based Testing Frameworks Integrated with Selenium WebDriver," International Scientific Journal of Engineering and Management, vol. 1, no. 5, Dec. 2022, [Online]. Available: https://doi.org/10.55041/ISJEM00087.

[14] B. García, F. Ricca, M. Leotta and M. Munoz-Organero, "Test automation with Selenium: A survey," Information and Software Technology, vol. 194, art. 108077, 2026, [Online]. Available: https://doi.org/10.1016/j.infsof.2026.108077.

[15] K. Lim, Y. Kwon and D. Kim, "A Longitudinal Study of Vulnerable Client-side Resources and Web Developers' Updating Behaviors," in Proceedings of the 2023 ACM on Internet Measurement Conference (IMC '23), pp. 162-180, Oct. 2023, [Online]. Available: https://doi.org/10.1145/3618257.3624804.

APPENDIX

The Appendix contains the scripts and sections of programming languages code used in the study.

```
{
  "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",
    "chai": "^5.1.0",
    "moment": "^2.29.4",
    "chromedriver": "^144.0.0",
 "wdio-chromedriver-service": "^8.1.1",
 "wdio-video-reporter": "^4.0.5",
 "@moroo/wdio-slack-reporter":"^8.0.1",
  },
  "scripts": {
    "test": "wdio run ./wdio.conf.ts"
  },
}
```

Listing A1: Package.json.

```
import video from 'wdio-video-
reporter';
import moment from 'moment';

export const config = {
reporters: ['spec',
```

```
        [ video, {
          saveAllVideos:
false,
          videoSlowdownMultiplier:
10,
        }],
    ],

afterTest: async function (test,
context, { error, result, duration,
passed }) {
        if (error &&
browser.capabilities.browserName ===
'chrome') {
        const timestamp =
moment().format('YYYYMMDD_HHmmss')
              await
browser.saveScreenshot(`./errorscree
n/error_${timestamp}.png`);
        }
    }
}
```

Listing A2: JavaScript code.