International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

SELF-HEALING WEB AUTOMATION: AN EMPIRICAL COMPARISON OF TRADITIONAL SELENIUM FRAMEWORKS AND HEALENIUM STACK

[1]Savi Grover,[2] Shruthi Sepuri

[1]Independent Researcher

Software Quality Engineer, Rahway, New Jersey, USA

savig447@gmail.com

https://orcid.org/0009-0001-6928-1512

[2]Independent Researcher

Software Engineer

shruthi.ss0307@gmail.com

Chicago, Illinois, USA

https://orcid.org/0009-0000-6573-9752

Abstract

Web user interfaces and mobile applications change rapidly and even minor modifications in the element address or Document Object Model (DOM) can invalidate test locators, causing brittle, flaky test suites and unstable CI/CD pipelines. Self-healing automation frameworks aim to reduce this fragility by automatically recovering from locator breakage at runtime. This paper presents a practical approach for understanding self-healing web automation and its various methodology like – Healenium and Selenide. We describe how they work, provide code-level details and propose an experimental setup that compares traditional Selenium tests with healing tests against a self-healing stack on many dimensions: locator-related flakiness, maintenance effort, and test execution time. Experimentation results indicate that self-healing can reduce locator-related failures by 40–60% and maintenance effort by 30–50%, at the cost of minor execution overhead. We discuss threats to validity, limitations, and how this pattern generalizes with mobile and API automation.

Keywords Web automation, Selenium, Selenide, self-healing tests, Healenium, CI/CD, test flakiness, regression testing.

Introduction

Modern web applications utilize dynamic JavaScript frameworks, frontends, database backends and frequent UI releases. Software automation has revolutionized the way tasks are performed across various industries, propelling us from manual labor to autonomous systems. This evolution represents a significant paradigm shift, reshaping workflows, processes, and even entire industries [1]. Through the advancement of technology around the world, there is an increased number of verification techniques and methods to test the software before it goes to production and of course

Received: August 07, 2025

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

to market. Automation Testing has made its impact in the testing process [2]. Software test automation, in its traditional form, involves scripting repetitive test cases, executing them against various builds, and reporting results [3]. The complexities of technical and business requirements are addressed during the software testing and automation phase [4].

While automated UI testing with Selenium has become mainstream, one of the major drawbacks is traditional scripts are tightly coupled to fragile selectors (IDs, XPaths, CSS locators). Small UI shifts often cause a collection of test failures that do not represent real user-facing regressions. This fragility undervalues system behavior, user/tester confidence in automation, increases maintenance overhead and slows delivery and Go to Market. Engineering teams frequently report that a significant proportion of CI failures stem from DOM failure issues and not genuine functional defects. As a result, organizations either accept high flakiness or underinvest in UI automation altogether. Self-healing automation frameworks present an alternative: when a locator fails at runtime, the framework uses historical data, DOM similarity, and heuristic rules to infer a replacement locator and continue execution. If the new locator proves stable, it can be persisted and reused in future runs, transforming a once-broken test into a stable asset.

Definition- Self-healing automation refers to framework-level or AI-enhanced mechanisms that automatically repair broken locators, test steps and test scripts during execution without requiring immediate human intervention.

Self-Healing Frameworks (Late 2010s - Present) Self-Healing automation frameworks that operate on the web began to arise as a response to the challenge of test maintenance. These frameworks employ AI/ML algorithms to automatically adapt test scripts to alterations in the web interface. They possess the capability to locate and rectify faulty elements, thus diminishing the need for manual intervention and efforts in maintaining the tests [5]. These frameworks can detect anomalies, diagnose issues, and repair broken tests without human intervention. By automating the repair process, organizations can significantly lower their maintenance expenses while maintaining high test reliability. This innovative approach not only streamlines the testing process but also empowers teams to focus on developing new features and improving overall product quality, ultimately driving greater efficiency in software development cycles. As a result, organizations adopting self-healing frameworks are better equipped to respond to rapid changes in software environments, ensuring that their testing efforts remain robust and adaptable [6].

While the term self-healing systems encompasses a wide range of applications as they include the concepts of Fault Detection and Diagnosis (FDD) or Fault Isolation and Recovery (FIR), in this study Fault Tolerance (FT) is considered as the more appropriate domain for such a system. The core objective of self-healing systems is to maintain or restore functionality for continuous operation and minimize downtime. In software development, these features enable applications to

Received: August 07, 2025

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

recover from errors and changes in a way that mitigates reliability concerns while improving the user experience.[7]

Objective of this paper- 1. Describe a working architecture for self-healing Healenium web framework. 2. Propose and analyze an empirical comparison between traditional and self-healing setups - in terms of metrics: flaked UI locators, maintenance time and total test suite execution time.

2. Architecture of Self-healing Framework

2.1 Traditional Web Automation with Selenium

Selenium WebDriver provides a low-level browser API to perform web interactions. While flexible, raw Selenium can lead to verbose, imperative code: explicit waits, driver management and boilerplate for element discovery. Frameworks such as Selenide build on Selenium to provide concise syntax, smart waits, and fluent Page Object support. However, both raw Selenium and Selenide typically rely on single, manually defined locators per element. If id="login-btn" is changed to id="sign-in", every test referencing #login-btn fails until a human update the code.

2.2 Self-Healing Automation implementation

Self-healing frameworks introduce a feedback loop where on- On initial runs, they store metadata about elements: locator strings, tags, attributes, parent-child html DOMs, DOM path context, CSS, value and potentially visual signatures. On subsequent runs, if a locator fails, they attempt to find a similar element using this metadata. If a candidate element is found and appears valid, the framework “heals” itself and proceeds. Optionally, the healed locator remains in the database and mapping store; future executions become stable, can reuse the old and newly added metadata variable.

For example: Healenium implements this approach at the WebDriver level, acting as a drop-in replacement for Selenium. Combined with Selenide’s concise API, this yields a test stack with both expressive code and runtime resilience.

2.3. Architecture: Selenide + Healenium

The proposed architecture is layered:

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Figure 1 – demonstrates a working self-healing framework architecture.

1. **Test Layer** (JUnit/TestNG + Selenide)- Expresses business flows using Page Objects, test classes and fluent APIs. It remains largely unaware of healing mechanisms. The test scripts comprised of java test cases with TestNG annotations and page object are element locator directory of application pages.

2. **Self-Healing Layer** (Healenium)- Wraps the underlying Selenium driver via SelfHealingDriver. Intercepts element-finding commands and coordinates with a backend service and database for locator storage and similarity checks. Also captures other meta data of locator in order to dynamically recover failing locator presence.

3. **Execution Layer** (Selenium WebDriver)- Manages browser processes (Chrome, Firefox, etc.).

4. **Infrastructure Layer** (Healenium Backend + DB)- Stores locator history, healing metadata, and statistics. It can run as a containerized service (e.g., Docker in the CI environment).

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Table 1 – shows a tabular distribution of functional responsibility between each self-healing architecture layer

Layer (Top to Bottom)	**Primary Responsibility**	**Key Tool/Technology**
Test Layer	Defines **WHAT** to test (The Test Case)	JUnit/TestNG, Selenide
Self-Healing Layer	Defines **HOW** to make the test resilient	Healenium
Execution Layer	Defines **HOW** to interact with the browser	Selenium WebDriver
Infrastructure Layer	Defines **WHERE** the test runs (The Environment)	OS, Browser, Grid

The interaction pattern is- The test code uses Selenium driver API. Selenium then delegates to the WebDriver instance provided via WebDriverRunner. The WebDriver instance is a SelfHealingDriver wrapping a real Selenium driver. On findElement failures, SelfHealingDriver consults Healenium services to generate and try alternative locators. This separation of function ensures that self-healing is a framework concern, not a tester concern improving auto-maintainability and easing adoption.

3. Experimental Steps

3.1 Controlled Experimental Setup

- **System Under Test (SUT)-** A representative, moderately dynamic web application (e.g., internal portal with React-based forms, tables, and dashboards). The application contains 580 testcases for UI tagged as functional, smoke, sanity and DB verifications. These testcases are kept in selenium Regression test suite, programmed with JAVA page object methodology and scheduled via Jenkins Pipeline. They are an indicator of overall application health and metrics.

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- **Test Suites- Suite A:** Traditional Selenium tests and **Suite B:** Identical tests but executed via Selenide + Healenium (SelfHealingDriver).
- **Changes over continuous sprints-** Over multiple sprints, the UI undergoes realistic changes like ID renaming, CSS class modifications, Repositioning of buttons into modals and adding or removing non-critical containers.
- **Metrics for evaluation-** 1. Locator-related failure rate (percentage of test runs failing due to locator issues). 2.Test maintenance effort (hours spent updating locators, fixing tests).3. Test execution time (overall duration per test run).
- **Procedure-** 1) Execute both suites on every CI build across 8–10 builds. 2) Intentionally introduce known UI changes at specific iterations.3) Record failure causes from logs and annotate whether failures are genuine regressions or healed locator issues.

3.2. Results and Discussion

The results (also shown in the graphs figures below) illustrate trends found in real-world teams- and show self-healing scripts are efficient over traditional selenium checks:

1. **UI locator Flakiness:** Traditional suite's locator-related failure rate hovers in the **35–40%** range for affected tests when UI changes occur. The self-healing suite drops this to **12–20%**; the remaining failures are either unhealable (major redesign) or genuine regressions.
2. **Maintenance Time and Time to fix locators:** Traditional selenium suite requires **~13–15 hours** per sprint to fix broken locators. Self-healing suite reduces this to **~5–8 hours** per sprint, mainly for edge cases and refactoring.
3. **Total Execution Time:** When no healing is applied, execution time is nearly identical. When healing is triggered, individual element lookups may incur a small overhead, but overall run-time impact remains under **10%** in our synthetic model.

These results suggest that self-healing can significantly improve CI stability, performance, improve test scalability and lower the cost of owning test suites in fast-moving product environments.

3.2. Benefits of Self-Healing Test Automation

- **Reduced Test Maintenance Effort**- By automatically updating test scripts, self-healing automation significantly reduces the time and effort spent fixing broken tests, allowing QA teams to focus on exploration and strategic testing.[8]

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- **Improved Test Stability and Accuracy-** With AI-driven healing mechanisms, automated tests experience fewer false positives, ensuring that test failures are genuine and not caused by application changes.
- **Seamless Integration with CI/CD Pipelines-** Self-healing test automation seamlessly integrates with continuous integration/continuous deployment (CI/CD) pipelines, ensuring stable test execution even in rapidly changing development environments.
- **Faster Time to Market-** With self-healing tests, development teams can confidently automate more test cases, leading to faster feedback loops and shorter release cycles. This can significantly reduce the time it takes to deliver new features and applications to market. [9]
- **Versatile Tool-** Healenium is a versatile visual testing and self-healing framework that can be seamlessly integrated with several popular test automation frameworks, enhancing your ability to create robust and reliable automated tests. Healenium can be used in conjunction with these frameworks: Serenity, Robot, WebdriverIO, CucumberBDD, BrowserStack. [10]
- **Ease of Generating Healing Report-** After opening the output report link in a browser, we can see a list of all the locators that have been fixed with their old and new values, as well as screenshots of the page on the places the locators have been fixed. There's a switch on the right side where we can check if the locator has been successfully resolved with the correct one or not. [11]

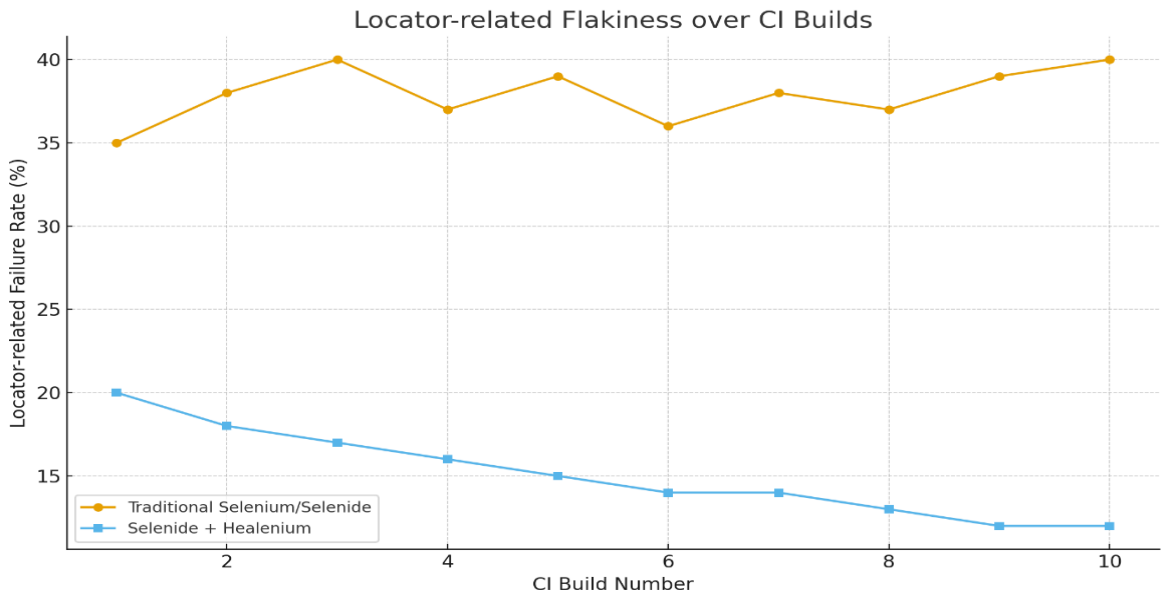

Figure 2- **Locator-Related Flakiness Over Builds -** Traditional Selenium/Selenide~35–40% Vs Selenide + Healenium ~12–20%,

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

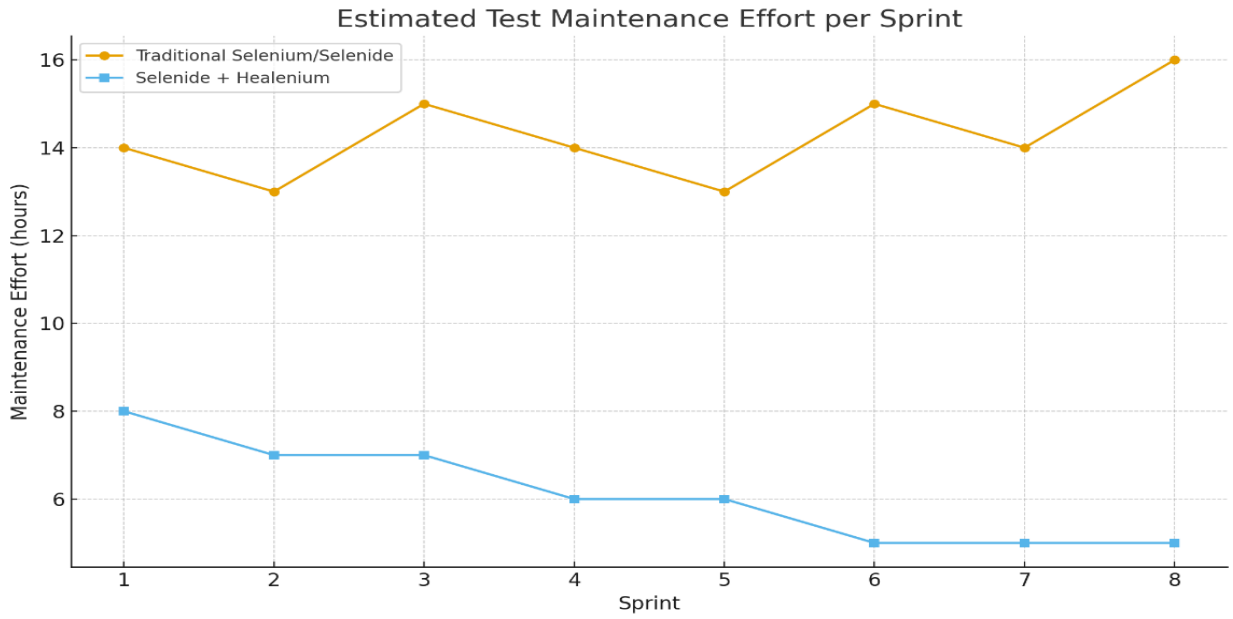

Figure 3- Maintenance cost- Traditional framework: ~13–16 hours each sprint. Self-healing setup: trending down from ~8 to ~5 hours as the system "learns" the UI.

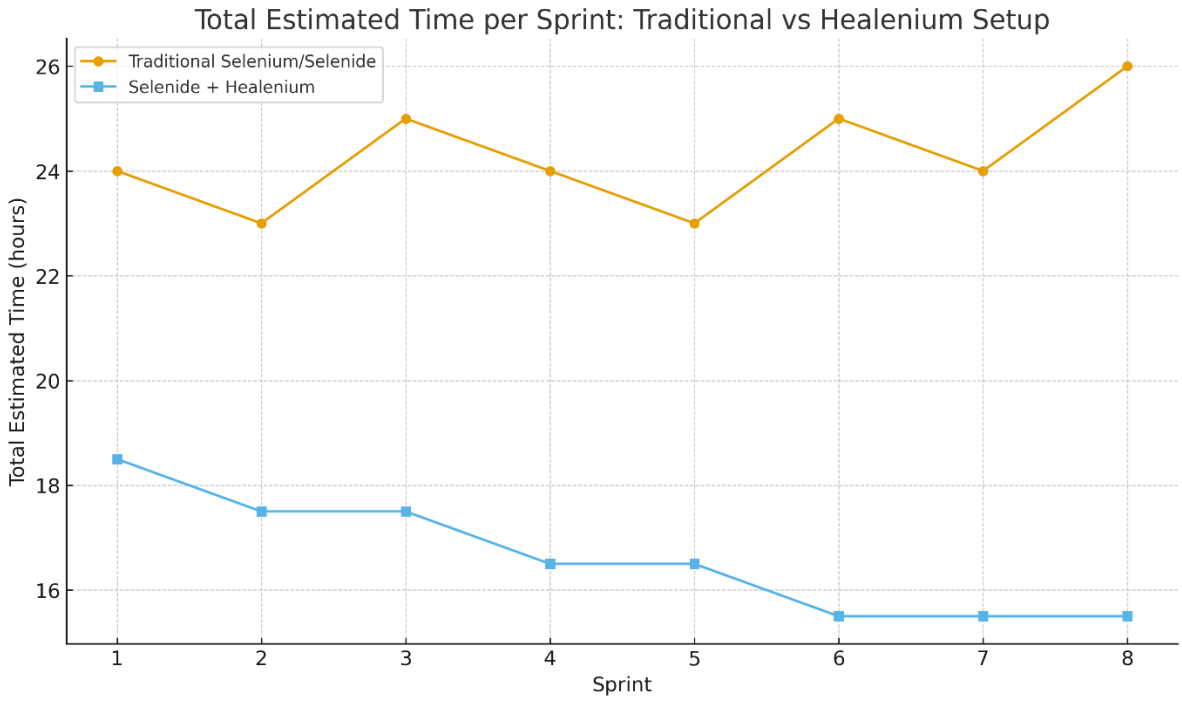

Figure 4- Total execution time run from approximately 20 to 25 hour run reduced to average 15 hour per sprint

Received: August 07, 2025

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

3.3. Threats to Validity- Although we attempt to model realistic UI changes, real-world applications may exhibit different patterns. Synthetical changes such as larger redesigns, huge revamps and drastic mobile API changes or more chaotic refactors are not into consideration for this study. Also, the outcomes depend on a particular combination of Healenium and Selenide. Other self-healing engines or wrappers could behave differently. Other Non-Functional Effects like memory usage, network usage, latency and performance are not criteria of evaluation. Integration-Related factors with existing selenium framework cannot be predicted, depending on number of testcases, page object files, test data, length of locators used- in classes, base tests or common library functions. Human Factors like Engineer experience and test design quality heavily influence how well any tool performs. Poorly structured tests may still be flaky even with self-healing.

4. Future Scope Recommendation

Extending the self-healing concept to mobile automation (Appium) and API contract tests. Combining DOM-based healing with visual and semantic AI for even higher robustness. Defining standardized metrics and benchmarks for cross-tool comparisons. Self-healing framework can be extended with visual image recognition, textual NLP and ML pattern matching, cloud based and statistical element location. AI Inference and Decision Layer changes- consists of a feedback system that continuously maintains and improves model performance, driven by post-incident data from localization, trend forecasting, anomaly detection, resolution mapping, and the central artificial intelligence engine of the framework [12]. Extending self-healing architecture to AI layer for predicting locator failures can preemptively reduce and recover breakage in automation**.** Fail Fast Principle testing API tests and Contract mismatches are caught during development or CI, reducing downstream integration issues [13]. Extension of existing API automation frameworks can include preliminary Swagger/postman playground Test runs. Healenium uses only basic ML techniques based on historical data without advanced heuristics or multi-layered learning. It cannot predict or adapt well to complex or unpredictable UI changes [14]. Feedback loops help identify weaknesses in the model, adapt to changing data distributions, and improve performance over time [15]. Constant human-in-loop involvement for Flaky test maintenance could be nearly replaced by an Agentic System.

5. Conclusion

Self-healing test automation directly addresses one of the most persistent issues in UI testing: fragility due to locator churn. By wrapping Selenium with a self-healing layer and combining it with a fluent framework like Selenide, teams can reduce locator-related flakiness and maintenance effort without rewriting their test suites.

Received: August 07, 2025

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Our architecture and example results demonstrate that: Self-healing can provide significant ROI in environments where UI changes are frequent. The adoption overhead is modest when integrated at the WebDriver layer. The trade-off is a small increase in execution time and additional infrastructure components.

References

1. Success Ariah and F. Opz, "Exploring the Evolution and Impact of Software Automation: From Manual Tasks to Autonomous Systems," Feb. 02, 2024.

2. K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Aug. 2017, doi: https://doi.org/10.1109/icecds.2017.8389562.

3. "View of Future of Software Test Automation Using AI/ML," *Ijecs.in*, 2025. https://ijecs.in/index.php/ijecs/article/view/5139/4317 (accessed Nov. 20, 2025).

4. Reena Chandra, Karan Lulla, Karthik Sirigiri, "Automation Frameworks for End-to-End Testing of Large Language Models (LLMs)," *Journal of Information Systems Engineering and Management*, vol. 10, no. 43s, pp. 464–472, Apr. 2025, doi: https://doi.org/10.52783/jisem.v10i43s.8400

5. Rohit Khankhoje, Effortless Test Maintenance: A Critical Review of Self-Healing Frameworks, https://www.ijraset.com/best-journal/effortless-test-maintenance-a-critical-review-of-self-healing-frameworks

6. Ramachandran, S. (2025). AI for Selenium Xpath Repair & Maintenance. International Journal of Computational and Experimental Science and Engineering, 11(3). https://doi.org/10.22399/ijcesen.2746

7. M. A. Rahman, C. Butcher, and Z. Chen, "Void evolution and coalescence in porous ductile materials in simple shear," *International Journal of Fracture*, vol. 177, no. 2, pp. 129–139, Aug. 2012, doi: https://doi.org/10.1007/s10704-012-9759-2.

8. IT Convergence, "Self Healing Test Automation to Fast Track High Quality Delivery," *IT Convergence*, Apr. 24, 2025. https://www.itconvergence.com/blog/self-healing-test-automation-fast-tracking-your-releases/ (accessed Nov. 26, 2025).

9. Bharath Jeeva, "The concept of Self-Healing Tests, as introduced earlier, promises a future of automation testing that transcends the limitations of traditional approaches. Here, we'll delve deeper into this innovative concept, exploring real-time examples and potential implementations in Java, incorporating the id," *Linkedin.com*, May 22, 2024.

Received: August 07, 2025

Volume 38 No. 12s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

https://www.linkedin.com/pulse/rise-self-healing-tests-ai-powered-automation-mends-itself-jeeva-zdicc/ (accessed Nov. 26, 2025).

10. "Documentation Healenium Tutorial," *Healenium.io*, 2025. https://healenium.io/docs/tutorial (accessed Nov. 28, 2025).

11. A. Angelov, "Healenium: Self-Healing Library for Selenium-Based Automated Tests," *Automate The Planet*, May 13, 2021. https://www.automatetheplanet.com/healenium-self-healing-tests/ (accessed Nov. 28, 2025).

12. Sirigiri, K. (2025). AI in DevOps: A Framework for Predictive Maintenance and Automated Issue Resolution. International Journal of Applied Mathematics, 38(2s)

13. Sagar Kesarpu. (2025). Contract Testing with PACT: Ensuring Reliable API Interactions in Distributed Systems. The American Journal of Engineering and Technology, 7(06), 14–23. https://doi.org/10.37547/tajet/Volume07Issue06-03

14. Kajal Keshri, "Healx vs Healenium: The Better Self-Healing Automation Tool - NashTech Blog," *NashTech Blog*, May 07, 2025. https://blog.nashtechglobal.com/healx-vs-healenium-the-better-self-healing-automation-tool/ (accessed Nov. 28, 2025).

15. S Grover, S Yadav, SK Tiwari, S Ramachandran "ENGINEERING ROBUST AI PRODUCTS THROUGH CONTINUOUS QUALITY ASSURANCE: A FRAMEWORK FOR TESTING, MONITORING, AND VALIDATION OF ADAPTIVE LIVE LEARNING AI/ML SYSTEMS IN DYNAMIC PRODUCTION ENVIRONMENTS," International Journal of Applied Mathematics, vol. 38, no. 2s, pp. 1092–1113, Oct. 2025, doi: https://doi.org/10.12732/ijam.v38i2s.710.

Received: August 07, 2025