

# AI-powered software testing tools: Full autonomy remains a distant goal

Vahid Garousi

Queen's University Belfast, UK  
Azerbaijan Technical University (AzTU), Azerbaijan

Nithin Joy

British Telecom PLC, UK

Davide Taibi

University of Oulu, Finland

## Abstract:

AI-powered software testing tools have taken test automation to a new level, promising improved efficiency, reduced maintenance effort of test-automation code, and possibly even enhanced defect-detection. In this paper, we systematically review and classify the set of 56 AI-assisted software testing tools available on the market (as of 2024).

Our review shows that AI-powered software testing tools can provide benefits to software test engineers, by potentially increasing effectiveness and efficiency of their software testing tasks, from test planning, to test-case design, to test execution and test-code maintenance. However, there are limitations and possibly even issues in these tools—such as false positives, and lack of domain (contextual) knowledge. A human test engineer should still peer review and validate any testing artifacts, e.g., test cases and test code, generated by the AI tools. For years to come, we predict that humans and machines (software testers and AI-powered testing tools) have to still work together. The road for full autonomy of these tools is still long.

**Keywords:** Software testing, artificial intelligence, generative AI, systematic tool review

## 1 INTRODUCTION

Software testing is effort-intensive for human software test engineers, and any help from non-AI and also AI-powered software testing tools is welcome by test engineers, for increasing effectiveness and efficiency of their software testing tasks.

[1]Advances in machine learning (ML), natural language processing (NLP), and computer vision have enabled the emergence of AI-powered testing tools, transforming traditional testing practices. These tools automate key testing tasks, including self-healing test automation, AI-driven test and test-code generation, and visual regression testing. By leveraging AI, modern testing tools aim to address longstanding challenges, such as the high manual effort required for test-case design, test-code maintenance, flaky tests, and UI inconsistencies, ultimately enhancing efficiency, accuracy, and scalability in software testing [2, 3].

As highlighted by a Gartner report in 2024 [4], in 2023 15% of software companies integrated AI testing tools into their software engineering practices, while they forecast a 80% of adoption in 2027. With such emergence of many recent AI-powered testing tools, it is timely to review these tools, explore their strength and limitations. That is the goal of this paper.

## 2 PROCEDURE FOR THE SYSTEMATIC TOOL REVIEW

Our goal was to systematically review and compare the available AI testing tools on the market. We utilized the established guidelines for performing systematic multivocal reviews [5] in software engineering. Our systematic tool review aimed at finding out and understanding:

- (1) the AI-Assisted testing automation tools available on the market
- (2) their features, and possible improvements in effectiveness and efficiency of software testing
- (3) their limitations

Using the systematic review guidelines, we performed the searches in the Google search engine. Our search string format was: "AI-X software testing tools", where "X" was replaced with: "based", "assisted", "powered", "augmented", and "enabled". These are common terms used for testing tools, in which AI is an enabler.

Only tools whose product sheets (web pages) explicitly mentioned using AI were included. General-purpose software testing tools without AI-powered features were not included in the pool of tools to be reviewed.

After completing the systematic search process and applying the above inclusion / exclusion criteria, 56 AI-based testing tools were selected for review and analysis.

All extracted data was systematically recorded in a Google spreadsheet, serving as a structured repository for the list of tools, data extraction, tool categorization and analysis. The online spreadsheet can be found in [bit.ly/AI\\_testing\\_tools](https://bit.ly/AI_testing_tools).

### 3 LIST OF THE AI-POWERED TESTING TOOLS AND CLASSIFICATIONS OF THEIR AI FEATURES

Among the 56 AI-powered testing tools reviewed in our study and reported in Table 1, we identified five different categories of AI features, as discussed below.

#### 3.1 AI-powered test generation

This feature enable the automated creation of test cases and/or automated test code, and is available in 43 tools (77%). These tools leverage additional input sources, including code analysis (identifying high-risk areas through static or dynamic analysis), requirements and specifications (using Natural Language Processing to extract test scenarios from documentation), user behavior analytics (generating tests based on real user interactions), test coverage gaps (targeting under-tested code areas), and configuration data (creating tests for different environments, devices, and system settings).

By integrating these diverse data sources, AI-powered testing tools can provide large test suites for a given System Under Test (SUT) in a few seconds or minutes, a task which could have taken an experienced software test engineer hours or days. In our selected tools, we noticed different approaches to AI-powered test generation:

- low (or no) code approach for test generation using AI-based NLP (20 tools);
- low (or no) code approach using record / playback (9 tools);
- AI-based analysis of code-coverage (13 tools);
- other approaches such as AI-powered model-based testing (3 tools).

A concrete example from the two no-code (codeless) approaches are shown in Figure 1. For example, a tester could write, *"Verify that the login button redirects the user to the dashboard after entering valid credentials"* and the AI-NLP tool would translate this into a test script compatible with automation frameworks like Selenium or Appium.

Test generation using AI-based code-coverage analysis is another widely offered feature, which identifies untested or high-risk areas in code (e.g., modules with high cyclomatic complexity) and automatically generates targeted test cases to improve coverage (offered by tools such as DiffbBue Cover, and TestRigor). BaseRock tool claims that it can provide *"up to 80% code coverage without developer intervention"*.

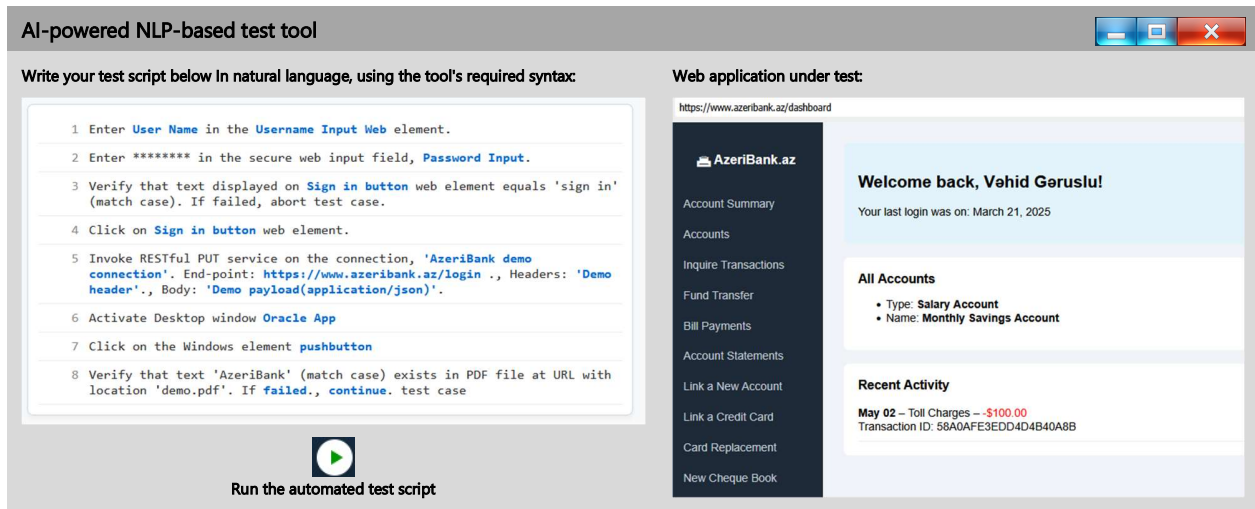


Figure 1- A conceptual example of the AI-powered NLP-based no-code approaches for test-case generation (the idea is from the following video: [bit.ly/ACCELO-AI-codeless-testing](https://bit.ly/ACCELO-AI-codeless-testing)).

### 3.2 AI-powered self-healing of UI test-code

AI-powered self-healing of UI test-code enables automated adaptation (repair) of (broken) test scripts when UI elements change, reducing test maintenance costs for test engineers and improving test stability (32 tools - 57%).

In conventional automated testing, scripts rely on predefined locators such as IDs, XPath, or CSS selectors to interact with UI elements. However, when the underlying code or design of the application changes (e.g., a developer renames a button's ID), these locators can break, causing test failures. Self-healing tools, often powered by AI and machine learning, can automatically recognize when an element locator breaks and intelligently find alternative locators by analyzing patterns like attributes, element hierarchy, or visual similarity. They then update the script automatically, reducing maintenance efforts and minimizing disruptions in the testing process.

For example, consider an automated test script that clicks a "Submit" button on a HTML page, defined as: `<button id="submit-btn">Submit</button>`. The button was initially identified using the following Java Selenium code snippet, using ID locator: `WebElement submitButton = driver.findElement(By.id("submit-btn"));` and `submitButton.click();`. Later, a front-end developer of the SUT changes the button's ID to: `<button id="send-btn">Submit</button>`. In traditional test automation, this change would cause the test script to fail because it cannot find the element with `id="submit-btn"` anymore. With **self-healing automation**, the AI tool can detect that the original locator no longer works but can identify the button by other attributes, such as its text content ("Submit") or position on the page. It automatically updates the script with the new locator or adapts the strategy (e.g., switching from an ID-based locator to a text-based one), allowing the test to continue running without any tester's intervention.

### 3.3 AI-powered visual testing

AI-powered visual testing (validation) can detect visual inconsistencies or regressions in User Interface (UI). These regressions could be unexpected changes or errors in how the UI looks when the application is viewed on different screen resolutions (e.g., mobile, tablet, or desktop), or in various browsers (e.g., Chrome, Firefox, Safari). For example, if a button is misaligned on a mobile screen or an image doesn't display properly on Firefox, AI can automatically detect these issues during testing, without requiring manual inspection by a tester. This feature is offered by 16 tools (29%).

### 3.4 AI-powered test-result and failure analysis

AI-powered test-result and failure analysis features automate root cause identification, reducing debugging time for testers (11 tools - 20%). These tools leverage machine learning and pattern recognition to analyze logs, execution results, and

historical defect data. By detecting recurring failure patterns, AI can suggest probable causes, categorize failures (e.g., flaky tests vs. real defects), and recommend fixes (tools such as Appsurify TestBrain, and Copado Robotic Testing).

Some tools apply AI-based clustering to group similar failures, helping testers prioritize issues efficiently (tools such as Testify Pulse, and ReportPortal). Others integrate with CI/CD pipelines to provide real-time diagnostics, reducing release delays (tools such as Tricentis Tosca, and LoadMill). AI can also analyze operational logs to detect performance bottlenecks and API failures (tools such as Parasoft SOAtest, and TestResults).

### **3.5 Other AI-powered features**

23 tools provide other AI features. Some offer AI-driven test prioritization to focus testing on high-risk areas (two example tools: Avo Automation, Testify Pulse), while others use predictive analytics to anticipate failures before they occur (Copado Robotic Testing, LoadMill).

AI-based exploratory testing is also offered (Keysight Eggplant Test), and automated test-data generation can create realistic test inputs (Virtuoso, Avo Automation).



LoadMill	x	x		x					User behavior (operational profile, logs)			x	
Mabl	x	x				x							
Machinet AI Unit Tests (a plugin for IntelliJ)	x				--AI describes what each test method is doing --AI generates mocks, objects, and non-trivial assert statements --AI decides whether to use mocks or not.			x		x			
Parasoft JTest	x							x		x			
Parasoft Selenic		x		x									
Parasoft SOAtest	x					x			API logs (recorded traffic)		x		
pCloudy	x	x	x		Observability agent, for full testing visibility	x						x	
Perfecto		x											
ReportPortal				x									
retest	x		x				x					x	
SauceLabs	x	x	x	x		x						x	
Smartbear - VisualTest			x		AI-powered object recognition								
Smartbear - TestComplete		x											
Sofy Co-Pilot	x		x	x	--Hyper-intelligent (optimized) regression testing --Possibility of further training of the built-in AI model	x						mobile apps	
taskade	x												x
Tabnine	x									x			
Test Grid	x	x		x		x							
Testify - Pulse	x			x	--AI-hierarchical clustering: groups API behaviors based on similarities, identifying both documented and undocumented patterns. It ensures each pattern is tested, improving coverage and detecting edge cases. --AI dependency analysis: Maps system component relationships for better test generation			x			x		
TestingWhiz	x	x	x				x						
TestResults		x	x		--AI Virtual user tests the software as a user would. --Visual hints: AI-based spatial algorithm to detect interaction points without predefined hints, enabling adaptive and robust automated testing.								
TestRigor	x	x				x							
TestSigma		x				x							
Tricentis Testim	x	x			--Learns user flows, recognizes repeated sequences and suggests reusable elements. --Captures screenshot of every test step and compares to previous screenshots to identify what has been changed, to suggest appropriate test cases. --AI helps in the development of well-architected, clean tests that optimize reuse and minimize maintenance.		x					x	
Tricentis Tosca	x	x	x		--A feature named RiskAI analyzes changes in SAP systems to identify high-risk areas needing prioritized testing, reducing release scope while achieving risk coverage	x	x						x
Unit-test.dev	x									x			
Virtuoso	x	x			Test data generation	x						web apps	
Webo.ai (by Webomates)	x	x					x					web apps	
Workik	x							x		x			

## 4 BENEFITS OF AI FEATURES: INCREASING EFFECTIVENESS AND EFFICIENCY OF TESTING

The main reason for development and inclusion of AI features in these tools is to support test engineers, and for increasing effectiveness and efficiency of testing. To assess the benefits of AI features, we used various data sources. A key source of data was [www.g2.com](https://www.g2.com), a widely recognized platform for software reviews. G2.com states on its homepage: *"Find the right software and services based on 2,917,200+ real reviews"*. For most of the tools under review, we found extensive practitioner reviews on G2.com discussing both benefits and limitations of the tools. Other data sources for us were the webpages and online documentation of the tools, vendor reports, and the large number of online demo videos for each tool. On all the qualitative (descriptive) data were collected, we used qualitative coding, to cluster and synthesize the benefits and limitations of AI features. Note that we did not conduct our own empirical studies to assess the benefits and limitations of AI features.

### 4.1 Potential increase in test effectiveness using AI features

AI-powered software testing tools could enhance test effectiveness by improving defect detection, expanding test coverage, and reducing human bias in test selection. Unlike test efficiency (which focuses on reducing effort and time), test effectiveness is about how well testing detects defects, ensures software reliability, and increases confidence in the system under test (SUT).

Through qualitative analysis of tool documentation and user feedback, we have identified key ways in which AI-generated test artifacts could contribute to improving test effectiveness.

- **More comprehensive test coverage (reducing testing gaps):** AI-generated test artifacts could expand coverage by analyzing code structure, system behavior, and historical defects to identify test gaps that manual testers might overlook (example tools: Diffblue Cover, TestRigor).
- **Higher defect detection rate (catching more bugs):** AI-powered testing could improve defect detection by dynamically analyzing software behavior, identifying edge cases, and detecting flaky tests that manual or script-based testing might fail to catch (example tools: AppliTools, Mabl).
- **Making test coverage more objective (reducing human bias):** AI-powered testing could objectively assess risk by analyzing historical failure data, system logs, and real-world usage patterns, ensuring that test cases align with actual risk rather than human intuition (example tools: Functionize, Tricentis Tosca).
- **Prioritizing critical and risky areas of the system:** AI-powered testing could intelligently prioritize high-risk areas, ensuring that business-critical or frequently changed components receive more intensive testing than stable, low-risk areas (example tools: Qodo, Testim).
- **Detecting defects earlier in the development cycle:** AI-powered testing could analyze code changes in real-time and suggest targeted tests to detect potential defects immediately after code commits, reducing defect leakage to later stages (example tools: LoadMill, ReportPortal).
- **Ensuring human oversight of AI-generated test artifacts:** AI cannot yet fully replace human judgment, but it could significantly assist in reducing testing gaps, improving defect detection rates, and making test coverage more structured and data-driven.

### 4.2 Potential increase in test efficiency using AI features

AI-powered software testing tools could enhance test efficiency by reducing manual effort, accelerating test execution, and optimizing resource usage. Unlike test effectiveness (which focuses on improving defect detection and test quality), test efficiency improvements relate to reducing the time, effort, and cost of achieving a given level of testing quality.

We identified five ways in which AI-generated test artifacts could contribute to increasing test efficiency.

- **Faster test execution with AI optimization.** AI-powered tools could dynamically analyze test results, historical data, and system behavior to optimize test execution sequences. Instead of running all tests equally, AI can prioritize the fastest and most relevant tests, significantly reducing overall execution time (example tools: LoadMill, ReportPortal).



- **Reduction in manual test maintenance effort.** Maintaining traditional automated test scripts is time-consuming, as small application changes often break test cases. AI-powered self-healing test automation could reduce maintenance time by automatically updating broken test scripts when UI elements change (example tools: Testsigma, Tricentis Tosca).
- **Minimization of test redundancy and resource waste.** AI-powered test optimization could eliminate redundant or duplicate tests by analyzing past execution data and removing unnecessary test cases that do not contribute to defect detection. This prevents running excessive tests and optimizes test execution time and computing resources (example tools: Functionize, Parasoft SOAtest).
- **Reduction of human effort in test case design.** AI-generated test artifacts could automate repetitive aspects of test creation, such as generating test scripts for various configurations and scenarios. This could reduce the burden on test engineers, enabling them to focus on high-value exploratory testing and analysis (example tools: Qodo, Testim).
- **Seamless CI/CD integration.** This would ensure that only necessary tests run for each code change, minimizing test cycle time. This could accelerate release cycles without increasing defect risk (example tools: AppliTools, Mabl).

An important point to consider in this context is ensuring human oversight of AI-generated test artifacts. While AI-generated test artifacts could greatly improve test efficiency, they always require human oversight to ensure that optimizations do not compromise test quality. Over-reliance on AI-driven test selection without validation could lead to gaps in coverage or missed defects.

## 5 LIMITATIONS OF AI-POWERED TESTING FEATURES

While AI-powered testing tools offer numerous advantages, they also present certain limitations that organizations must consider. We would ideally expect that these limitations would decrease as the technology and tools are improved.

- **Lack of explainability and transparency.** Many AI algorithms operate as "black boxes," providing little insight into their decision-making processes, which can hinder trust and make it difficult for testers to validate AI-generated outcomes. A tester using AppliTools stated, "*While visual AI works great, sometimes I don't understand why it marks something as a failure when it shouldn't be.*" Similarly, users of Mabl have noted that its self-healing mechanism sometimes applies fixes without explaining why, making debugging challenging.
- **Maintenance and continuous learning requirements.** AI-powered testing tools require ongoing updates and retraining to keep pace with evolving software applications. A user of Testsigma remarked, "Keeping AI models up-to-date and maintaining automated tests requires ongoing effort; the tool sometimes applies outdated learning when left unattended." Similarly, Tricentis Tosca has been reported to struggle with adapting to frequent UI changes unless retrained. This maintenance demand can offset the efficiency gains promised by AI automation. (qamadness.com)
- **Complexity in implementation and integration.** Integrating AI-powered tools into existing testing workflows can be challenging. Users of QMetry have found that, "Adopting AI testing required process overhauls and additional team training, making it a slower transition than expected." Similarly, an ACCELQ user noted, "The AI's automation logic requires extensive setup before it becomes useful." This complexity can deter teams from adopting AI-based testing, particularly when dealing with legacy systems. (code-intelligence.com)
- **Risk of overfitting and false positives.** A Diffblue Cover user highlighted, "The AI-generated test cases sometimes pass redundant assertions, flagging expected changes as issues." Similarly, ReTest users reported, "The tool frequently detects UI differences that are not real defects, making it harder to focus on actual issues." This high rate of false positives can reduce efficiency instead of improving it. (testfort.com)
- **Ethical and bias concerns.** Biases from the training data can lead to skewed test prioritization. A tester using AppliTools noted, "Some visual AI tests failed disproportionately on certain interface styles, suggesting an underlying bias in its learned data." Similarly, a Functionize user stated, "AI seemed to prioritize some test cases over others without clear reasoning, which raised concerns over fairness in coverage." Addressing these biases is critical to ensuring balanced and reliable test automation. (code-intelligence.com)
- **Limited creativity and handling of edge cases.** AI-based test case generation is less effective in identifying unpredictable user behaviors and complex business logic. A Testim user commented, "The tool works well for common flows, but it doesn't think outside the box to explore unexpected user actions." Similarly, testers using



Mabl mentioned that "It automates regression tests effectively, but exploratory testing still needs human creativity." (qamadness.com)

- **User control over AI model training.** Some AI-powered testing tools do not allow to train or fine-tune their AI models, instead relying on vendor-controlled updates. A Testim user stated, "We wish we had more control over training the AI for our specific use cases, rather than waiting for vendor improvements." Similarly, in Tricentis Tosca, the AI-powered self-healing updates are applied automatically by the vendor, which limits customization for unique project needs.

## 6 RECOMMENDATIONS FOR TOOL IMPROVEMENT

To address these challenges and increase the adoption of AI-powered testing, future tool development should focus on:

- **Enhancing explainability.** Providing clear justifications for AI-driven decisions to build tester confidence and enable debugging of AI-generated artifacts.
- **Reducing false positives.** Improving AI's ability to differentiate actual defects from non-critical UI or functional changes to prevent alert fatigue.
- **Simplifying integration.** Offering better API support, detailed documentation, and more customizable workflows to allow smoother integration into CI/CD pipelines.
- **Allowing user-controlled AI training.** Enabling test engineers to fine-tune AI models for their specific needs rather than relying entirely on vendor-controlled updates.
- **Mitigating bias.** Developing AI that dynamically adjusts to changing applications and diverse test conditions, rather than favoring repetitive patterns in its dataset.

### 6.1 Future research directions

While AI-powered testing tools continue to evolve, further empirical studies are needed to assess their real-world impact, effectiveness, and challenges. Future research should explore how these tools perform in various domains and project sizes, helping to establish best practices for when and how AI-assisted testing should be applied.

Additional key research directions include:

- **Hybrid AI-human testing.** Combining AI's automation with human exploratory testing, allowing AI to suggest tests while testers refine them.
- **Adaptive AI models.** Developing AI that learns continuously from test execution feedback, reducing the need for frequent manual retraining.
- **Ethical AI testing standards.** Establishing guidelines to ensure that AI-powered testing tools do not reinforce systemic biases in test prioritization, defect prediction, or code analysis. This includes developing bias detection mechanisms in AI models and designing AI-assisted testing approaches that promote fairness, diversity, and inclusion.
- **Industry-wide benchmarking.** Defining standardized evaluation criteria for AI-powered tools to help practitioners compare effectiveness, reliability, and usability across platforms.

## ACKNOWLEDGEMENT

This work was partially funded from the HIVEMIND project funded by the European Union under Grant Agreement nr. 101189745 and by the Business Finland project 6GSoft.

## REFERENCES

- [1] M. Last, A. Kandel, and H. Bunke, *Artificial intelligence methods in software testing*. World Scientific, 2004.
- [2] A. Sahoo, *AI-Infused Test Automation: Revolutionizing Software Testing through Artificial Intelligence*. OrangeBooks Publication, 2023.
- [3] M. Winteringham, *Software Testing with Generative AI*. Manning, 2024.
- [4] Gartner Research, "Market Guide for AI-Augmented Software-Testing Tools," 2024. [Online]. Available: <https://www.mabl.com/blog/recognizing-7-years-of-ai-innovation-in-test-automation>.

This is the post-print of a paper that has been published with the following DOI: [doi.org/10.1109/MS.2025.3557895](https://doi.org/10.1109/MS.2025.3557895)

---

- [5] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, pp. 101-121, 2019.

#### **AUTHOR BIOGRAPHIES:**

Vahid Garousi is a Professor at Queen's University Belfast, UK, a Visiting Professor at Azerbaijan Technical University, and an international Software Testing Strategist and Consultant. His areas of expertise include software testing and empirical software engineering. Since 1998, he has helped many software companies improve their software testing practices and processes. Contact: [v.garousi@qub.ac.uk](mailto:v.garousi@qub.ac.uk)

Nithin Joy is a Software Engineer with BT Group, UK. His areas of expertise include software testing and modern software engineering practices. Contact him at [nithin.joy@bt.com](mailto:nithin.joy@bt.com)

Davide Taibi is a Professor at the University of Oulu, Finland, and a Software Architecture Consultant. His research focuses on software architecture quality and refactoring. He helps companies select and adopt cloud-native technologies while modernizing their applications. Contact: [davide@taibi.it](mailto:davide@taibi.it)