ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.02 No.04 (2024)

SOFTWARE AUTOMATION TESTING: COMPARING NO-CODE, LOW-CODE, AND TRADITIONAL APPROACHES (2025)

Shawaiz Arif
Email: shawaizarif1@gmail.com
Department of Information Technology, Superior University, Lahore, Pakistan
Muhammad Faisal
Email: faisalshafiq02@outlook.com
Department of Computer Science, Superior University, Lahore, Pakistan
Muhammad Saad Khan Lodhi
Email: skpro571@gmail.com
Department of Computer Science, Superior University, Lahore, Pakistan
Supervisor Prof. Saleem Zubair Ahmad
MS. Software Engineering
Saleem.zubair@superior.edu.pk
Department of Software Engineering, Superior University, Lahore, Pakistan
Co-Supervisor Sabah Arif
Sabah.Arif@Superior.edu.pk
Department of Software Engineering, Superior University, Lahore, Pakistan

Abstract:
Automated software testing tools play a vital part in guaranteeing the trustability, effectiveness, and quality of software operations. As the lifecycles of software development grow more intricate, the demand for effective robotization testing has risen vastly. This paper offers a relative evaluation of low- code, no- code, and traditional automation testing tools, assessing them grounded on essential criteria similar as customization, client support, user-friendliness, and integration capabilities..
Furthermore, it explores the advantages and limitations of each tool category, providing a comprehensive guide for new testers to select the most appropriate automation technique based on their skill level and project requirements. The findings indicate that no-code tools offer accessibility for non-technical testers, low-code tools balance ease of use with moderate customization, and traditional tools provide maximum flexibility and scalability for experienced testers handling complex projects. By understanding the strengths and weaknesses of each category, testers and organizations can make informed decisions to optimize their software testing strategies [2].

Keywords:
No-Code Tools, Low-Code Tools, Traditional Automation Tools, Software Quality Assurance (SQA), Test Automation, Agile Testing, Scalability, Customization, Ease of Use, Implementation Speed, Cost-Effectiveness, Complex Testing Scenarios, AI Integration, Machine Learning, Automation Tools Comparison, Software Testing, Continuous Integration, Testing Efficiency, Automation Strategy, Software Development Lifecycle, Real-World Case Studies.

I. INTRODUCTION

Software testing has develop an important phase in the software development lifecycle (SDLC), particularly with the rise of Agile methodologies and DevOps practices. Automation testing tools have reduce the time and effort involved in testing, therefore accelerating the software release cycle. These tools are distributed into three main types low- code, no- code, and traditional tools, each catering to different user conditions. For new testers, the challenge

.

lies in choosing the best tool grounded on their programming proficiency and project demands.

As software testing becomes more integral to continuous integration (CI) and continuous delivery (CD) pipelines, understanding the nuances of these automation tools is crucial. This paper aims to provide an in-depth comparison of the three categories of automation tools and offer guidance for new testers on how to select the most appropriate tool for their skills and the complexity of their testing requirements [5].

II. Literature Review

Automation testing has importantly evolved over the past decade, driven by the need for rapid software development cycles, Agile methodologies, and DevOps practices. Experimenters have significantly analyzed automation testing tools, comparing their effectiveness, scalability, and ease of use to identify their strengths and limitations.

2.1 No-Code Automation Testing Tools

No-code automation tools have gained traction due to their user-friendly interfaces and ability to democratize test automation. Smith et al. (2021) highlight that these tools enable non-technical testers to automate test cases without programming expertise, making them highly suitable for organizations seeking rapid automation with minimal learning curves. The study further emphasizes that Selenium IDE, a widely used no-code tool, simplifies test creation through record-and-playback features, but its limitations include a lack of flexibility and reduced support for complex testing scenarios.

Cypress, another emerging tool in this space, introduces a hybrid approach, incorporating low-code elements while remaining accessible to users with limited technical knowledge. However, Johnson and Lee (2020) argue that no-code tools struggle with scalability and require integration with other frameworks for enterprise-level applications.

2.2 Low-Code Automation Testing Tools

Low-code automation tools serve as a bridge between no-code simplicity and traditional automation flexibility. Kumar (2022) compared Katalon Studio, Ranorex, and Ghost Inspector with traditional testing frameworks and found that low-code tools significantly reduce the learning curve while offering more control over test scripts. These tools often support both UI-based test creation and script-based modifications, providing a balanced approach for testers with moderate coding experience.

One notable advantage of low-code tools is their ability to integrate with Continuous Integration/Continuous Deployment (CI/CD) pipelines while maintaining ease of use. Research by Subramaniam (2020) suggests that organizations adopting low-code automation frameworks experience faster test case development cycles and improved test maintenance efficiency. However, a key drawback remains their limited customization when dealing with highly complex, enterprise-level applications requiring deep scripting capabilities.

2.3 Traditional Automation Testing Tools

Traditional automation tools, such as Selenium WebDriver, Appium, and JMeter, remain the most powerful solutions for advanced test automation. Johnson and Lee (2020) highlight that these tools offer unparalleled flexibility, customization, and scalability, making them the preferred choice for complex testing environments. They allow testers to write detailed, reusable scripts using programming languages such as Java, Python, and C#, enabling in-depth validation of software functionalities.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

Despite their advantages, traditional tools have a steep learning curve and require extensive coding knowledge. A study by Brown (2019) indicates that while organizations using Selenium WebDriver benefit from its open-source nature and vast community support, the framework's initial setup and maintenance efforts can be resource-intensive. Moreover, Kumar (2022) notes that traditional tools require robust test management practices, as poorly structured scripts can lead to high maintenance costs and increased debugging efforts.

2.4 Comparative Studies and Industry Trends

Several comparative studies have examined the effectiveness of different automation testing approaches. A study conducted by Smith and Brown (2021) found that while no-code tools accelerate test case creation, their limited flexibility makes them suitable only for simple scenarios. In contrast, low-code tools provide a balance between usability and customization, whereas traditional tools excel in scalability and advanced scripting capabilities.
Additionally, with the growing relinquishment of AI- driven testing, recent exploration proposes that hybrid automation models incorporating both low- code and traditional frameworks are getting progressively popular. Organizations are using AI- powered automation tools that enhance self- healing capabilities, predictive analytics, and intelligent test generation, further optimizing the effectiveness of automation testing strategies(Subramaniam, 2020).

III. RESEARCH METHODOLOGY

This study employs a organized comparative analysis approach to assess the effectiveness of low-code, no-code, and traditional automation testing tools. The methodology contains of the following steps:

1. **Selection of Tools:** A diverse set of tools from each category was carefully chosen based on their industry relevance, adoption rate, and functional capabilities. Selection criteria included factors such as market demand, user reviews, and compatibility with modern software development methodologies.

2. **Evaluation Criteria:** The tools were measured based on numerous key factors, including customization, ease of use, scalability, customer support, integration capabilities, reporting features, and cost-effectiveness. These factors were selected to provide a comprehensive understanding of each tool's strengths and weaknesses.
3. **Data Collection:** Data was gathered from multiple reliable sources, including official tool documentation, industry whitepapers, peer-reviewed research papers, case studies, user forums, and direct feedback from software testing professionals. This multi-source approach ensured a well-rounded and unbiased assessment.
4. **Analysis and Comparison:** The collected data was systematically analyzed and presented in tabular format to highlight feature differences and comparative advantages among low-code, no-code, and traditional automation tools. The analysis focused on practical usability, efficiency, and real-world application in testing environments.
5. **Validation and Expert Review:** The findings were validated through real-world case studies, direct testing of selected tools, and consultation with experienced software testers and industry experts. Their insights provided additional credibility and practical relevance to the study's conclusions.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

IV. Comparative Analysis

A. Low-Code Automation Testing Tools

Feature	**Ghost Inspector**	**Ranorex**	**Katalon Studio**
Customization	Moderate	High	High
Customer Support	Yes	Yes	Yes
End-to-End Testing	Yes	Yes	Yes
Ease of Use	High	Moderate	High
Free or Open Source	No	No	Partially
Trial Time	14 Days	30 Days	Free Tier Available
Platform Support	Web	Desktop, Web, Mobile	Web, Mobile, API
Language Support	JavaScript	C#, VB.NET	Java, Groovy
Data-Driven Testing	Yes	Yes	Yes
Reports	Yes	Yes	Yes
Test Case Development	Drag & Drop	Code + UI	UI & Scripting
Slack/Email Integration	Yes	Yes	Yes
Test Pass/Fail Alerts	Yes	Yes	Yes

Key Insight: Low-code tools like Katalon Studio and Ranorex provide a balance of ease of use and customization. While they are suitable for teams with limited coding knowledge, they allow for more flexibility than no-code tools. Ghost Inspector stands out as a tool with an easy-to-use interface but more limited customization capabilities (Smith & Brown, 2021).

B. No-Code Automation Testing Tools

Feature	**Selenium IDE**	**Cypress**	**TestComplete**
Customization	Low	Moderate	High
Customer Support	Yes	Yes	Yes
End-to-End Testing	Yes	Yes	Yes
Ease of Use	Very High	High	High
Free or Open Source	Yes	No	No
Trial Time	Free	Free Tier	30 Days
Platform Support	Web	Web	Web, Desktop, Mobile
Language Support	JavaScript	JavaScript	JavaScript, Python
Data-Driven Testing	No	Yes	Yes
Reports	No	Yes	Yes
Test Case Development	Record & Playback	UI + Code	UI & Scripting

ISSN E: 3006-1466
ISSN P: 3006-1458
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW

Slack/Email Integration	No	Yes	Yes
Test Pass/Fail Alerts	No	Yes	Yes

Key Insight:
Selenium IDE and Cypress are excellent tools for testers without programming skills. While Selenium IDE is ideal for simple scenarios with record-and-playback, Cypress introduces more flexibility with coding, but it requires some JavaScript knowledge. TestComplete provides a more professional no-code solution with rich features like data-driven testing and integrated reporting (Johnson & Lee, 2020).

C. Traditional Automation Testing Tools

Feature	**Selenium WebDriver**	**Appium**	**Postman**	**JMeter**	**SoapUI**	**Cucumber**
Customization	Very High	High	High	High	High	High
Customer Support	Community	Community	Yes	Yes	Yes	Yes
End-to-End Testing	Yes	Yes	Yes	Yes	Yes	Yes
Ease of Use	Low	Moderate	High	High	Moderate	High
Free or Open Source	Yes	Yes	Yes	Yes	Yes	Yes
Trial Time	Free	Free	Free	Free	Free	Free
Platform Support	Web	Mobile	API	Load Testing	API	BDD Testing
Language Support	Java, Python, C#	Java, Python, C#	JavaScript	Java	Java, Groovy	Java
Data-Driven Testing	Yes	Yes	Yes	Yes	Yes	No
Reports	No	No	Yes	Yes	Yes	Yes
Test Case Development	Code-Based	Code-Based	UI + Code	UI & Scripting	UI & Scripting	BDD
Slack/Email Integration	No	No	Yes	Yes	Yes	Yes
Test Pass/Fail Alerts	No	No	Yes	Yes	Yes	Yes

Key Insight: Traditional tools like Selenium WebDriver and Appium give high customization and flexibility, making them ideal for complex, large- scale systems that require extensive configuration and detailed scripting. These tools demand further specialized expertise but offer unparalleled control and scalability. Cucumber and Postman are particularly effective for BDD (Behavior- Driven Development) testing and API testing, respectively (Kumar, 2022).

V. Advantages and Disadvantages

A. No-Code Automation Testing Tools

No-code automation tools are designed for testers with little to no programming experience. These tools provide a user-friendly interface that allows users to create and execute automated tests without writing code.

1. Advantages

- Ease of Use – No-code tools offer a visual, drag and-drop interface, enabling quick test creation without requiring programming skills.
- Rapid Test Development – Test automation is significantly accelerated, making it an ideal choice for Agile and DevOps environments.
- Accessibility for Non-Technical Testers – Business analysts, product managers, and manual testers can contribute to automation efforts.
- Low Maintenance Effort – The record-and-playback approach reduces the need for script maintenance.
- Seamless Collaboration – Non-technical stakeholders can actively participate in test automation.

2. Disadvantages

- Limited Customization – No-code tools may not support complex test logic or integrations with advanced testing frameworks.
- Scalability Constraints – These tools may struggle to support large-scale automation projects.
- Vendor Lock-In – Functionality is restricted to the features provided by the vendor, limiting extensibility.
- Integration Challenges – Some no-code tools have limited support for CI/CD pipelines and version control systems.
- Potential Cost Implications – Enterprise-grade no-code automation tools often require expensive licensing.

B. Low-Code Automation Testing Tools

Low-code automation tools provide a hybrid approach, combining graphical test case creation with scripting capabilities. These tools cater to testers with basic programming knowledge who require more flexibility than no-code solutions.

1. Advantages

- Balance of Usability and Customization – Low-code tools offer both visual interfaces and scripting support, making them adaptable for various testing needs.
- Faster Test Development – Pre-built automation frameworks and reusable components reduce the time required for test creation.

- Improved Integration with DevOps Pipelines – Most low-code tools support CI/CD and version control systems.
- Reduced Learning Curve – Compared to traditional tools, low-code solutions require minimal coding knowledge.
- Enhanced Maintainability – Automated tests can be updated efficiently with minimal script modifications.

2. Disadvantages

- Limited Control Over Advanced Test Scenarios – While more flexible than no-code tools, low-code solutions may not provide full customization.
- Higher Costs Compared to Open-Source Tools – Many low-code platforms require paid subscriptions, increasing project expenses.
- Tool-Specific Constraints – Feature availability depends on the vendor, potentially leading to compatibility issues.
- Dependency on UI-Based Elements – Test cases may be affected by UI changes, requiring frequent updates.

C. Traditional Automation Testing Tools

Traditional automation testing tools, such as Selenium WebDriver, Appium, and JMeter, provide full scripting capabilities, enabling advanced test automation. These tools are best suited for experienced testers with programming proficiency.

1. Advantages

- High Customization and Flexibility – Traditional tools allow testers to create complex automation scripts tailored to specific requirements.
- Scalability for Large-Scale Applications – Suitable for enterprise-level projects requiring extensive automation.
- Multi-Language Support – Supports programming languages such as Java, Python, and C#, providing flexibility in test development.
- Open-Source Availability – Many traditional tools are free, reducing licensing costs.
- Robust Community Support – Large user communities contribute to continuous improvements, plugins, and troubleshooting resources.
- Seamless Integration with DevOps Pipelines – Well-suited for CI/CD environments and Agile workflows.

2. Disadvantages

- Steep Learning Curve – Requires programming knowledge, making it less accessible for non-technical testers.
- Time-Intensive Test Development – Writing and maintaining automation scripts demands significant effort.
- Complex Initial Setup – Configuration and infrastructure setup require additional expertise and resources.
- High Maintenance Costs – Test scripts must be continuously updated to adapt to application changes.
- Resource-Intensive – Traditional automation necessitates dedicated automation engineers, increasing operational costs.

VI. Recommendations for New Testers

For new testers looking to choose the best automation tool:

1. **No Coding Knowledge:**

Recommended Tool: No-code tools such as Selenium IDE or TestComplete.
Why: These tools offer the simplest user interface and allow non-technical testers to automate basic tasks quickly (Johnson & Lee, 2020).

2. **Basic Programming Skills:**
    - **Recommended Tool:** Low-code tools like Katalon Studio.
    - **Why:** These tools offer a balance of simplicity and customization, allowing testers with some technical expertise to create effective automated tests.
3. **Experienced Testers:**
    - **Recommended Tool:** Traditional tools such as Selenium WebDriver or Appium.
    - **Why:** These tools are best for testers with advanced programming knowledge who require complete control over the testing process (Kumar, 2022).

VII. Conclusion

The selection of an automation testing tool is a critical decision that significantly impacts the efficiency, effectiveness, and scalability of software testing efforts. This paper has explored the three primary categories of automation testing tools—low-code, no-code, and traditional tools—each of which offers unique benefits and challenges, depending on the tester's technical expertise and the nature of the project.

For non-technical testers or those with limited programming experience, no-code tools like Selenium IDE and TestComplete offer the quickest and easiest path to automate testing tasks, enabling them to perform essential test automation without needing to write a single line of code. These tools empower teams to implement rapid test automation, making them ideal for smaller projects or organizations looking to onboard testers with minimal technical expertise (Smith & Brown, 2021). However, their limited customization options may restrict their applicability for more complex or large-scale testing needs.

For testers who possess a basic understanding of programming, low-code tools such as Katalon Studio provide a middle ground. These tools combine the simplicity of no-code tools with the flexibility of traditional tools, offering a more customizable approach to automation testing without overwhelming testers with complex coding requirements. Low-code tools enable faster test development and are well-suited for smaller teams looking to scale their testing efforts while maintaining control over test execution and reporting (Kumar, 2022). However, their limitations in terms of deep customization and flexibility might make them unsuitable for highly complex or enterprise-level automation.

On the other hand, traditional automation tools like Selenium WebDriver and Appium are ideal for seasoned testers with advanced programming skills who require a high level of control over test development. These tools offer unmatched flexibility, scalability, and integration capabilities, making them the preferred choice for large enterprises or projects that involve complex testing scenarios. The trade-off, however, is the steep learning curve and time investment required to master these tools. Traditional tools provide the best solution for long-term, large-scale test automation projects, where customization, scalability, and integration with other development tools are paramount (Johnson & Lee, 2020).

Ultimately, the decision on which tool to choose depends on several factors, including the tester's skill level, the complexity of the test cases, and the project's size and long-term goals. For new testers, the key is to align their choice with their current technical proficiency while considering the project's scalability and maintenance needs. As automation tools continue to evolve, it is important for testers to stay updated on advancements in this space, ensuring that they can choose the best tool for their specific context and future-proof their testing strategies.

In conclusion, while no-code and low-code tools democratize automation testing by making it accessible to a broader audience, traditional tools remain indispensable for projects that require deep customization, high scalability, and extensive control over the testing process. A thorough understanding of the strengths and limitations of each category will empower testers to make informed decisions that best suit their project needs and organizational goals.

VIII. References

[1] Smith, J., & Brown, L. (2021). *Adoption of No-Code Automation Tools in Agile Environments. Journal of Software Testing, 15(2), 45-60.*

[2] Johnson, K., & Lee, M. (2020). *Comparative Study of Traditional and Modern Automation Testing Frameworks. Software Engineering Review, 28(1), 75-92.*

[3] Kumar, R. (2022). *Low-Code vs. Traditional Testing: Efficiency and Limitations. International Journal of Software Engineering, 30(3), 123-140.*

[4] Subramaniam, P. (2020). *The Evolution of Automation Testing Tools. Software Testing Practices, 12(1), 32-45.*

[5] Brown, A. (2019). *Selecting the Right Testing Tool: A Comparative Approach. Software Automation Journal, 24(3), 99-114.*

[6] Meszaros, G. (2007). *xUnit Test Patterns: Refactoring Test Code. Addison-Wesley.*

[7] Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective. Addison-Wesley.*

[8] Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley.*

[9] Koomen, T., & Pol, M. (1999). *Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing. Addison-Wesley.*

[10] Fewster, M., & Graham, D. (1999). *Software Test Automation: Effective Use of Test Execution Tools. Addison-Wesley.*

[11] Lee, K., Park, S., & Kim, J. (2020*). "No-Code and Low-Code Platforms: A Paradigm Shift in Automation*

[12] Lopez, A., Garcia, T., & Wang, Z. (2020). "*Challenges and Opportunities in Automating Software* Testing." *International Journal of Software Quality*.

[13] Martin, H., Taylor, J., & Schwartz, B. (2021*). "The Impact of Automation on Software Testing Efficiency." Software Quality Journal.*

[14] Smith, J., & Johnson, R. (2019). *"The Role of Traditional Automation Tools in Software Testing." International Conference on Software Testing.*

[15] Williams, A., & Zhang, H. (2022). "*Cost-Effectiveness of Low-Code and No-Code Tools for QA Teams." Journal of Information Technology and Management.*