Evaluating AI-Based Unit Testing Tools: A Focus on Usability, Reliability, and Integration

Hafsa Aarifeen
Department of Industrial Management
University of Kelaniya
Sri Lanka
hafsama-im19042@stu.kln.ac.lk

Dilani Wickramaarachchi
Department of Industrial Management
University of Kelaniya
Sri Lanka
dilani@kln.ac.lk

2025 5th International Conference on Advanced Research in Computing (ICARC) | 979-8-3315-3098-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICARC64760.2025.10963099

***Abstract*—This study presents an empirical comparison of Artificial Intelligence (AI)-based unit testing tools, focusing on their usability, reliability, and the challenges of integrating them into existing software testing frameworks. As AI-driven testing tools emerge as solutions to the limitations of traditional manual testing, particularly within agile methodologies and continuous integration/continuous deployment (CI/CD) pipelines, their effectiveness remains an area of active research. The research evaluates the potential of these tools to automate test generation and execution, thereby improving test coverage and bug detection while minimizing manual intervention. However, significant usability challenges persist, as developers often find these tools difficult to configure and deploy, especially in projects with varying complexities. Additionally, concerns about the reliability of AI-generated tests, particularly regarding consistent bug detection across different environments, are addressed. Integration into legacy systems poses further obstacles, as many organizations hesitate to overhaul established testing frameworks. The findings of this study offer insights into the current state of AI-based unit testing tools and provide recommendations for overcoming barriers to their widespread adoption in real-world development environments.**

***Keywords*—artificial intelligence, reliability, unit testing tools, usability, integration**

I. Introduction

AI-based unit testing tools face several challenges related to usability, reliability, integration, and their role in Continuous Integration/Continuous Deployment (CI/CD) environments. Usability issues arise from the complexity of AI algorithms and the need for high-quality training data, which can hinder user adoption and effective implementation [1], [2]. Reliability concerns stem from algorithmic biases and the potential for errors in AI-generated test cases, necessitating robust validation processes to ensure accuracy [3], [4]. Integration into CI/CD pipelines presents additional hurdles, including the need for seamless collaboration between AI tools and existing workflows, as well as managing the computational costs associated with deploying large language models [3], [5]. Furthermore, organizations must cultivate a supportive culture and provide structured training to navigate these challenges effectively, ensuring that AI enhances testing efficiency and software quality [2], [4]. This research aims to evaluate the usability, reliability, and integration potential of AI-based testing tools in real-world software development settings. The study will explore the tools' effectiveness in diverse project types, such as small, large, and complex systems including web applications and machine learning models. By addressing these challenges, the research seeks to offer valuable insights for both academic knowledge and industry practices, providing guidance on how AI testing tools can be successfully integrated into existing software practices..

The key research questions guiding this study are:

1) How do AI-based unit testing tools differ in usability, particularly in terms of developer adoption and ease of integration?

2) What is the reliability of these tools in detecting bugs and minimizing false positives across varied software environments?

3) How seamlessly can AI-based tools be integrated into existing CI/CD pipelines without requiring significant modifications?

By addressing these questions, this study aims to provide a comprehensive analysis of AI-based unit testing tools and their potential to transform software testing practices.

II. Related Works

A. Artificial Intelligence in Software Unit Testing

AI has transformed software unit testing by automating tasks like test generation, prioritization, and execution. Techniques such as machine learning (ML), deep learning (DL), and natural language processing (NLP) enhance error prediction, test case generation, and defect detection. For example, Adaboost effectively identifies change-prone classes, while NLP aids in converting natural language requirements into test constraints, improving coverage and accuracy [6], [7]. Models like GPT-4 further streamline testing by autonomously generating and refining unit tests [8].

Optimization techniques, such as genetic algorithms, maximize code coverage while minimizing test suite sizes, enhancing efficiency [9]. Despite these advancements, challenges

persist, including inconsistent performance across environments due to a lack of standardized datasets and data quality dependencies [7], [10].

AI tools like EVOSUITE and ATHENATEST showcase the potential of machine learning and deep learning in achieving higher coverage and reduced testing time compared to manual methods [11], [12]. However, challenges such as tool readability, integration, and performance variability remain barriers to widespread adoption. For instance, GitHub Copilot performs better with certain languages like Java and C++ than Python or Rust, while specialized tools like JUTAI demonstrate higher accuracy in specific domains but face limitations in edge cases [13], [14].

A hybrid approach combining AI-driven tests with manual oversight is often recommended to address limitations in data quality, standardization, and edge-case handling. Future advancements in data robustness and algorithm refinement are expected to further improve AI's effectiveness in real-world testing scenarios.

B. Usability, Reliability, and Integration of AI-Based Software Unit Testing Tools

AI-based unit testing tools are transforming software development by automating tasks like test case generation, execution, and adaptation. These tools improve efficiency, accuracy, and coverage, making them increasingly popular. Despite their potential, challenges persist in usability, reliability, and integration with existing testing frameworks.

1) Usability: AI-based software testing tools have the potential to significantly enhance usability by automating repetitive tasks and improving test coverage [15]. Tools like Testim and Functionize have been shown to reduce time-to-market and enhance software quality [16]. However, the steep learning curve and challenges related to accessibility, particularly for smaller teams, remain obstacles. To address these, frameworks like HUIA emphasize automation [17], while tools like DALAI integrate various functionalities to simplify processes [18]. There is a need for tools to provide more control to users, particularly developers using programming assistants like GitHub Copilot [19].

2) Reliability: AI-based software unit testing tools face challenges in reliability, especially in handling complex test cases. Optimization techniques such as multi-objective evolutionary algorithms (MOEAs) help improve test case generation [20]. However, gaps in test coverage, particularly for intricate scenarios, persist [21]. AI algorithm biases and lack of transparency hinder reliability, but techniques like metamorphic and classification-based testing are suggested to mitigate these issues [22]. Further empirical validation is essential to enhance consistency and accuracy [23].

3) Integration: Integrating AI-based tools into existing test suites and CI/CD pipelines is a critical concern. Tools like DIFFBLUE COVER and Randoop improve test coverage, but challenges remain when integrating with legacy systems [24]. Successful case studies from companies like Alibaba and IBM show the potential of AI in CI/CD, though legacy constraints hinder seamless adoption [1]. A strategic approach that balances AI and traditional tools is crucial for effective integration [25].

Key research gaps in AI-based software unit testing include a lack of comparative studies on popular tools such as EvoSuite, Randoop, and Diffblue Cover, particularly in usability, reliability, and integration. Usability issues like steep learning curves and interface complexity need more focus, and reliability concerns demand further empirical investigations into AI-generated test accuracy. Additionally, practical solutions for integrating AI tools into legacy systems and comprehensive guidelines for optimizing these tools in real-world environments are needed to maximize their adoption and effectiveness. AI-based software unit testing tools offer promising solutions to automate time-consuming tasks, improve testing accuracy, and enhance overall software quality. However, their full potential is yet to be realized, particularly concerning usability, reliability, and integration. To achieve widespread adoption and maximize the benefits of these tools, further research is needed to address gaps in comparative studies, usability challenges, reliability issues, and integration into existing workflows. By focusing on these areas, AI-based testing tools can become more accessible, reliable, and effective in modern software development environments.

III. METHODOLOGY

This study presents a systematic framework for evaluating AI-based unit testing tools, focusing on usability, reliability, and integration. While the analysis is centered on EvoSuite and Diffblue Cover, the evaluation methodology is designed to be broadly applicable, providing a template for assessing other tools in diverse contexts. By using standardized metrics and controlled experiments, this approach ensures that the findings can inform both academic research and industry practices.

To address scalability and generalizability, the framework supports future extensions to large-scale projects with complex dependencies and CI/CD pipelines. This research also proposes a hybrid approach that combines AI-driven test generation with human validation, paving the way for more adaptable and effective testing solutions beyond current tools.

A. Scope Definition and Tool Selection

1) Tool Selection: The selection of AI-based unit testing tools for this research was driven by the need to evaluate both academic and industry-relevant solutions. After a detailed comparison of available tools, two were chosen based on their unique features and alignment with the research objectives of usability, reliability, and integration, as shown in TABLE I.

TABLE I: COMPARISON OF AI-BASED UNIT TESTING TOOLS

Tool Name	AI-Based Unit Testing Exists	Industry Usage	Research Relevance	Variety of AI Approaches	Selected
EvoSuite	Yes	Widely adopted	High coverage and usability	Search-based algorithms	Yes
Diffblue Cover	Yes	Widely adopted	Strong CI/CD integration	Reinforcement learning	Yes
GitHub Copilot	Not focused on testing	Adopted	General-purpose tool	Limited to code generation	No
JUTAI	Yes	Low adoption	Limited by performance issues	Java-specific AI	No
AthenaTest	Yes	Low adoption	Focus on maintainability	Sequence learning	No
Mabl	Focuses on web testing	Adopted	Not relevant to unit testing	End-to-end testing focus	No
Randoop	Yes	Adopted	Random tests, low reliability	No advanced AI	No
Test.ai	Focuses on UI testing	Adopted	Not for unit testing	UI-specific algorithms	No
OpenAI Codex	General-purpose tool	Widely used	No structured test generation	General AI, not testing	No
Kex	Yes	Low adoption	Academic relevance	Symbolic AI techniques	No

EvoSuite was selected for its strong foundation in academic research and its use of advanced search-based algorithms for automated test generation. Its focus on achieving high coverage criteria, such as branch and line coverage, coupled with its scalability and support for both command-line and IDE plugin interfaces, makes it ideal for assessing reliability across diverse Java applications, from simple utilities to complex enterprise systems.

Diffblue Cover was chosen for its industry-focused approach, utilizing reinforcement learning to generate maintainable, high-coverage tests. Its widespread adoption in industry, particularly for Java applications, and its seamless integration into CI/CD pipelines make it a perfect candidate for evaluating practical usability and integration into real-world workflows.

The tools were compared based on their technical capabilities, usability, and real-world applicability within the scope of unit testing. GitHub Copilot and JUTAI, initially considered, were excluded from the study due to their limitations in generating robust and maintainable unit tests. GitHub Copilot, while effective as a general-purpose AI coding assistant, lacks specific optimization for structured unit test creation and exhibits reduced reliability with larger codebases. Similarly, JUTAI, though designed for Java-specific testing, struggles with scalability, integration into modern CI/CD workflows, and support for diverse programming environments, making it less suitable for comprehensive unit testing evaluations.

2) Scope Definition: The scope of this research is defined by the evaluation criteria used to assess EvoSuite and Diffblue Cover, specifically focusing on usability, reliability, and integration. These aspects were selected based on insights from prior literature, as outlined in TABLE II, to ensure the study addresses key practical considerations in the adoption of AI-driven unit testing tools.

TABLE II: JUSTIFICATION FOR SCOPE DEFINITION

Reference	Usability	Reliability	Integration	Performance	Accuracy	Adaptability	Documentation	Support
Abdallah et al., 2018	✓	✓	✓	✓	✓	✓	✓	✓
Alcon et al., 2021		✓	✓					✓
Amalfitano et al., 2023	✓	✓	✓				✓	
Amarasekara, 2023			✓					
Au et al., 2008	✓							
Bajaj & Samal, 2023	✓	✓	✓	✓	✓			
Baqar & Khanda, 2024		✓	✓					✓
Bolu, 2024			✓					
Fontes et al., 2021		✓	✓				✓	
Garousi et al., 2024	✓							
Gkikopouli & Bataa, 2023	✓							
Bolu, 2024			✓					
Dumay et al., 2022	✓	✓		✓				✓
Hussain et al., 2021		✓	✓		✓			
Miller et al., 2023	✓	✓		✓				
Molina et al., 2020	✓	✓	✓					
Hamed et al., 2022					✓			✓
Khan et al., 2023		✓		✓				
Ali et al., 2023	✓		✓					
Santos et al., 2023								✓
Tian et al., 2024		✓	✓					

Usability: Evaluates how easily developers can adopt and integrate the tools into their workflows.

Reliability: Assesses the consistency and accuracy of the tests generated by the tools.

Integration: Examines how seamlessly the tools integrate into existing development environments, particularly CI/CD pipelines and common testing frameworks.

B. Experimental Design and Configuration

The experimental design and configuration phase of this study aimed to evaluate the selected AI-based unit testing tools, EvoSuite and Diffblue Cover, in real-world scenarios. The tools were assessed based on their usability, reliability, and integration capabilities using small and medium-sized projects, which were selected to cover a range of software development scenarios and complexities.

1) Project Selection:

- The selected projects were categorized into small and medium-sized projects to assess the tools across various levels of complexity and dependencies.
- **Small projects** are defined as applications with straightforward requirements, limited functionality, and minimal dependencies. These projects typically require shorter development timelines and smaller teams and are referred to here as Small 1, 2, and 3.
- **Medium projects** are characterized by more comprehensive requirements, involving multiple interdependent modules and broader functional scopes. These projects typically have longer development timelines and require moderately sized teams to manage the increased complexity. They are referred to here as Medium 1, 2, and 3.

2) Evaluation Metrics: To assess the tools' effectiveness, the research focused on three main metrics: usability, reliability, and integration.

- **Usability** was measured using two metrics:

1) Time to first test
2) Error rate during setup

- **Reliability** was evaluated based on:

1) Bug detection rate
2) False positive rate
3) Test coverage

- **Integration** was assessed through the following:

1) Update time
2) Update coverage efficiency
3) Errors during updates

3) Test Environment: The experiments were conducted in a controlled environment to ensure consistency. Both tools were installed and executed on identical hardware and software configurations, eliminating external factors that could influence the results. The projects were run under identical conditions, ensuring fair and comparable results.

C. Execution and Data Collection

The Execution and Data Collection phase involved applying the selected AI-based unit testing tools, EvoSuite and Diffblue Cover, to evaluate their usability, reliability, and integration capabilities in real-world scenarios.

- Tool Setup and Environment:

Both tools were installed in a controlled environment to ensure consistency across experiments. The same hardware

configuration and standardized software environment (including dependencies, testing frameworks, and Java runtime) were used for all tools. Configuration adjustments were made as necessary to ensure compatibility with the selected projects.

- Running the Tools on Projects:

Each tool was applied to small, and medium projects, representing different levels of complexity. The tools generated unit tests, which were executed on the project codebases. The tests were run iteratively to assess consistency and reliability. The performance of the tools was evaluated by recording key metrics such as test generation time, success/failure rates, bug detection, and code coverage.

- Evaluation and Iteration:

Data were collected on the tools' performance in time efficiency, error rates, bug detection, and code coverage, with repeated experiments ensuring reliability. Deviations were analyzed for potential causes, providing a thorough and reproducible evaluation of usability, reliability, and integration.

IV. Results and Discussion

A. Evaluation of Tool Usability: EvoSuite vs. Diffblue Cover

- **Time to First Test**

The time taken to generate and execute the first unit test was measured for both tools across small and medium projects. As shown in TABLE III, Diffblue Cover demonstrated consistently faster setup and test generation times. On average, Diffblue Cover took 16.67 minutes for small projects and 28 minutes for medium projects, while EvoSuite took 22.33 minutes for small projects and 35 minutes for medium projects.

TABLE III: UPDATE TIME (IN MINUTES)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	20	22	25	32	38	35
Diffblue Cover	18	17	15	25	31	28

- **Errors During Setup**

The number of errors encountered during the setup process was also recorded to assess the ease of configuration for each tool. As presented in TABLE IV, EvoSuite generally exhibited fewer errors for small projects (average of 4.33 errors) compared to Diffblue Cover (4.67 errors). However, for medium projects, EvoSuite encountered more setup errors (average of 4.33 errors) compared to Diffblue Cover (6.33 errors).

TABLE IV: ERRORS DURING SETUP

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	3	5	5	3	3	7
Diffblue Cover	6	6	2	9	5	5

B. Evaluation of Tool Reliability: EvoSuite vs. Diffblue Cover

- **Bug Detection Rate**

EvoSuite demonstrated superior bug detection performance, with an average rate of 87.7% compared to Diffblue Cover's 83.5% as shown in TABLE V. EvoSuite consistently outperformed Diffblue Cover in identifying known bugs, particularly in medium projects. This highlights EvoSuite's reliability in detecting critical defects, especially in complex setups.

TABLE V: BUG DETECTION RATE (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	88	89	89	82	83	85
Diffblue Cover	90	89	84	81	77	80

- **False Positive Rate**

EvoSuite showed a lower false positive rate (average of 4.58%) than Diffblue Cover (6.76%) as shown in TABLE VI. Diffblue Cover struggled with higher false positives in medium projects, especially in Projects 1 and 2 (up to 10.14%). EvoSuite's lower false positive rate indicates greater accuracy in identifying actual bugs.

TABLE VI: FALSE POSITIVE RATE (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	3.02	2.57	4.78	4.56	5.32	7.25
Diffblue Cover	4.66	5.45	3.36	10.14	9.88	5.16

- **Test Coverage**

EvoSuite demonstrated more comprehensive test coverage across all metrics compared to Diffblue Cover. EvoSuite achieved an average of 91.5% line, 85.8% branch, and 88.2% method coverage, as shown in TABLE VII, TABLE VIII, and TABLE IX, respectively. In contrast, Diffblue Cover achieved 85.5% line, 79.5% branch, and 83% method coverage. EvoSuite's higher coverage demonstrates its effectiveness in testing critical code paths and edge cases.

TABLE VII: LINE COVERAGE (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	92	94	94	90	88	91
Diffblue Cover	90	86	88	83	84	82

TABLE VIII: BRANCH COVERAGE (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	86	86	91	86	81	85
Diffblue Cover	85	80	81	75	78	78

TABLE IX: METHOD COVERAGE (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	89	88	93	89	87	83
Diffblue Cover	85	86	85	78	81	83

C. Evaluation of Tool Integration: EvoSuite vs. Diffblue Cover

- **Update Time**

EvoSuite recorded an average update time of 2.67 minutes for small projects, slightly longer than Diffblue Cover's average of 2.33 minutes as shown in TABLE X. For medium projects, EvoSuite required 5 minutes on average, while Diffblue Cover took 4.67 minutes. Overall, both tools performed similarly in terms of update time, with Diffblue Cover marginally faster, indicating its slightly better efficiency in fast-paced environments.

TABLE X: UPDATE TIME (IN MINUTES)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	2	3	3	6	4	5
Diffblue Cover	2	3	2	5	4	5

- **Update Coverage Efficiency**

EvoSuite demonstrated a higher average update coverage efficiency of 88.33% for small projects, compared to Diffblue Cover's 87.67% as shown in TABLE XI. However, Diffblue Cover excelled in small project scenarios, while EvoSuite outperformed Diffblue Cover in medium projects, achieving an average coverage of 83%, compared to Diffblue Cover's 81.67%. EvoSuite exhibited more consistent coverage across medium projects, whereas Diffblue Cover showed better performance on smaller, less complex projects.

TABLE XI: UPDATED COVERAGE EFFICIENCY (in %)

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	88	89	88	82	84	83
Diffblue Cover	90	88	85	80	82	83

- **Errors During Updates**

EvoSuite recorded fewer errors during updates for small projects, with an average of 3.33 errors, compared to Diffblue Cover's average of 4.67 errors as shown in TABLE XII. However, EvoSuite experienced a higher error rate for medium projects, with an average of 7.33 errors, while Diffblue Cover showed a more stable error rate of 6 errors on average. This indicates that EvoSuite performs better in handling errors for smaller projects but faces challenges with more complex updates. Diffblue Cover, on the other hand, demonstrated a more stable error handling performance across project sizes.

TABLE XII: ERRORS DURING UPDATES

Tool	Small 1	Small 2	Small 3	Medium 1	Medium 2	Medium 3
EvoSuite	2	5	3	8	7	7
Diffblue Cover	5	5	4	5	5	8

D. Discussion

This study aimed to explore the usability, reliability, and integration potential of AI-based unit testing tools, specifically focusing on EvoSuite and Diffblue Cover. The results suggest several insights that align with the key research questions.

The first research question explored how AI-based unit testing tools differ in usability, particularly regarding developer adoption and ease of integration. The results indicate a clear usability distinction between EvoSuite and Diffblue Cover. Diffblue Cover stood out for its faster setup and test generation times, making it more attractive for developers working on smaller projects or in agile environments where speed is paramount. However, this speed comes with trade-offs, as Diffblue Cover exhibited a higher false positive rate, particularly for medium-sized projects, necessitating manual intervention. On the other hand, EvoSuite, though slower in setup, demonstrated more stable performance and greater reliability, making it a preferable choice for teams focused on ensuring thorough testing and reliability.

These findings highlight practical applications in industries like SaaS, where Diffblue Cover's speed accelerates development timelines, and sectors like healthcare or finance, where EvoSuite's rigorous testing ensures software quality and compliance.

Regarding the second research question, which addressed the reliability in detecting bugs and minimizing false positives, the study found that EvoSuite outperformed Diffblue Cover in bug detection rates, with a notable 87.7% detection rate. This superior reliability makes EvoSuite a strong candidate for use in critical applications, where comprehensive bug detection is essential. The higher reliability of EvoSuite and its ability to minimize false positives made it the preferred tool for complex projects requiring thorough testing. In contrast, while Diffblue Cover performed adequately in smaller projects, its decline in reliability for medium-sized projects highlighted the trade-off between speed and accuracy.

This dynamic may also apply to tools for other programming languages. The study's methodology can guide evaluations of Python, JavaScript, or other tools, helping adapt their strengths for varied use cases. For instance, tools with strong usability features, like Diffblue Cover, could be optimized for smaller-scale Python projects, while Java-focused tools like JUTAI could incorporate reliability enhancements seen in EvoSuite.

The third research question focused on the seamless integration of AI-based tools into existing CI/CD pipelines without requiring significant modifications. While both tools can be integrated into CI/CD pipelines, Diffblue Cover's faster setup time may offer an easier initial integration for smaller teams or projects where speed is prioritized. However, EvoSuite's more reliable and stable performance makes it a better fit for teams that can accommodate a slightly longer setup time in exchange for more comprehensive and stable testing.

In practical scenarios, EvoSuite might outperform Diffblue Cover in long-term projects with evolving requirements, where its stability ensures consistent test coverage over time. Conversely, Diffblue Cover can be invaluable in fast-paced industries such as e-commerce or digital marketing, where speed and adaptability are prioritized. This differentiation provides a clear roadmap for tool selection based on project needs and industry constraints.

A hybrid approach combining both tools could optimize testing for teams working on a mix of project sizes. Diffblue Cover could handle quick tests for smaller components, while EvoSuite would provide thorough testing for larger or more critical aspects.

V. CONCLUSION

This research evaluated two prominent AI-based unit testing tools, EvoSuite and Diffblue Cover, focusing on usability, reliability, and integration. The findings reveal distinct strengths for each tool, providing actionable guidance for their application in diverse software development scenarios.

EvoSuite excelled in providing comprehensive test coverage and detecting critical defects, making it particularly suited for projects requiring rigorous testing, such as those in healthcare, finance, or aerospace. Its ability to generate high-coverage tests and accurately identify bugs ensures software quality and robustness, making it an ideal choice for teams prioritizing thorough defect detection and reliability. To maximize its

effectiveness, organizations can invest in training and infrastructure improvements to manage its slower setup time and integration.

Conversely, Diffblue Cover demonstrated significant strengths in usability and integration, particularly in CI/CD workflows. Its faster setup and seamless integration into development environments make it advantageous for teams aiming to streamline testing in agile or fast-paced projects. This makes it well-suited for industries like e-commerce or SaaS, where rapid iteration and deployment are crucial. Teams using Diffblue Cover should complement it with manual validation or additional tools to mitigate its higher false positive rate, especially for medium-sized and complex projects.

Future research could explore hybrid approaches that leverage the strengths of both tools, combining Diffblue Cover's speed with EvoSuite's reliability to create a balanced testing strategy. Additionally, extending the evaluation to other programming languages and testing frameworks could provide broader insights, supporting the adoption of AI-based testing tools across diverse development environments.

In conclusion, the choice between EvoSuite and Diffblue Cover should be guided by project requirements. EvoSuite is ideal for complex systems that demand comprehensive, high-coverage testing, while Diffblue Cover is better suited for teams prioritizing efficiency, rapid deployment, and seamless CI/CD integration. By aligning tool selection with project goals, organizations can enhance testing effectiveness and software quality.

REFERENCES

[1] M. Baqar and R. Khanda, "The future of software testing: Ai-powered test case generation and validation," *arXiv preprint arXiv:2409.05808*, 2024.

[2] R. Khankhoje, "Ai in test automation: Overcoming challenges, embracing imperatives," *International Journal on Soft Computing Artificial Intelligence and Applications*, vol. 13, 02 2024.

[3] T. Chen, "Challenges and opportunities in integrating llms into continuous integration/continuous deployment (ci/cd) pipelines," in *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, pp. 364–367, IEEE, 2024.

[4] Y. Bajaj and M. K. Samal, "Accelerating software quality: Unleashing the power of generative ai for automated test-case generation and bug identification," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 7, 2023.

[5] A. Singh, O. Al-Azzam, *et al.*, "Artificial intelligence applied to software testing," in *CS & IT Conference Proceedings*, vol. 13, CS & IT Conference Proceedings, 2023.

[6] H. Hourani, A. Hammad, and M. Lafi, "The impact of artificial intelligence on software testing," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 565–570, IEEE, 2019.

[7] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, "Automatic generation of acceptance test cases from use case specifications: an nlp-based approach," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2020.

[8] V. Li and N. Doiron, "Prompting code interpreter to write better unit tests on quixbugs functions," *arXiv preprint arXiv:2310.00483*, 2023.

[9] A. Fontes, G. Gay, F. G. d. O. Neto, and R. Feldt, "Automated support for unit test generation: a tutorial book chapter," *arXiv preprint arXiv:2110.13575*, 2021.

[10] M. A. Job, "Automating and optimizing software testing using artificial intelligence techniques," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, 2021.

[11] J. M. Rojas, G. Fraser, and A. Arcuri, "Automated unit test generation during software development: A controlled experiment and think-aloud observations," in *Proceedings of the 2015 international symposium on software testing and analysis*, pp. 338–349, 2015.

[12] M. Tufano, D. Drain, A. Svyatkovskiy, S. K. Deng, and N. Sundaresan, "Unit test case generation with transformers and focal context," *arXiv preprint arXiv:2009.05617*, 2020.

[13] I. Siroš, D. Singelée, and B. Preneel, "Github copilot: the perfect code compleeter?," *arXiv preprint arXiv:2406.11326*, 2024.

[14] K. Kahur and J. Su, "Java unit testing with ai: An ai-driven prototype for unit test generation," 2023.

[15] F. Ricca, A. Marchetto, and A. Stocco, "Ai-based test automation: A grey literature analysis," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 263–270, IEEE, 2021.

[16] O. Vorochek and I. Solovei, "Research on artificial intelligence tools for automating the software testing process," *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies*, p. 58–64, Jul. 2024.

[17] F. T. Au, S. Baker, I. Warren, and G. Dobbie, "Automated usability testing framework," in *Proceedings of the ninth conference on Australasian user interface-Volume 76*, pp. 55–64, 2008.

[18] T. Räisänen, M. Lipsanen, and A. Föhr, "Ai powered tools for improving usability in digital archiving," in *Archiving Conference*, vol. 20, pp. 71–74, Society for Imaging Science and Technology, 2023.

[19] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of ai programming assistants: Successes and challenges," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–13, 2024.

[20] S. A. Abdallah, R. Moawad, and E. E. Fawzy, "An optimization approach for automated unit test generation tools using multi-objective evolutionary algorithms," *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 178–190, 2018.

[21] V. Garousi, N. Joy, and A. B. Keleş, "Ai-powered test automation tools: A systematic review and empirical evaluation," *arXiv preprint arXiv:2409.00411*, 2024.

[22] C. Tao, J. Gao, and T. Wang, "Testing and quality validation for ai software–perspectives, issues, and practices," *IEEE Access*, vol. 7, pp. 120164–120175, 2019.

[23] B. A. Yawer, J. Liss, and V. Berisha, "Reliability and validity of a widely-available ai tool for assessment of stress based on speech," *Scientific reports*, vol. 13, no. 1, p. 20224, 2023.

[24] M. Alcon, H. Tabani, J. Abella, and F. J. Cazorla, "Enabling unit testing of already-integrated ai software systems: The case of apollo for autonomous driving," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 426–433, IEEE, 2021.

[25] M. Gkikopouli and B. Bataa, "Empirical comparison between conventional and ai-based automated unit test generation tools in java," 2023.