Ranah Research
Journal of Multidisciplinary Research and Development
E-ISSN: 2655-0865
082170743613 ranahresearch@gmail.com https://jurnal.ranahresearch.com
DINASTI RESEARCH

DOI: https://doi.org/10.38035/rrj.v8i3

Generative AI-Based Software Testing Implementation Model in Agile Development Methodology: Systematic Literature Review

Linmas Ditasari[1], Teguh Raharjo[2]
[1]Faculty of Computer Science, Master of Information Technology, University of Indonesia, Jakarta, linmasditasari@gmail.com
[2]Faculty of Computer Science, Master of Information Technology, University of Indonesia, Jakarta

Corresponding Author: linmasditasari@gmail.com[1]

Abstract: The development of Artificial Intelligence (AI) technology, particularly Generative AI and Large Language Models (LLM), has had a significant impact on the software development process, including software testing activities. This study aims to conduct a Systematic Literature Review (SLR) to identify and analyze various Generative AI-based software testing frameworks applied in Agile development environments. The literature search process was conducted in several major scientific databases, namely IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Google Scholar with a publication period of 2021–2026. Based on the selection process using the PRISMA method, 35 articles were obtained that met the inclusion criteria. The analysis results show that most studies utilize Large Language Models (LLM), machine learning, generative models, and retrieval-augmented generation (RAG) techniques to support software test automation, such as test case generation, unit testing automation, and behavior-driven development testing. In addition, several studies also developed AI assistant and agentic AI-based frameworks that can be integrated into Agile development pipelines. This study found that the use of Generative AI can improve testing process efficiency, accelerate test case generation, and help improve software quality. However, several challenges remain, such as limited model accuracy, the need for high-quality datasets, and issues with the security and reliability of AI-based systems

Keyword: Generative AI, Software Testing, Agile Development, Large Language Models

INTRODUCTION

The advancement of information technology demands that organizations produce high-quality software in a short timeframe. To address this need, many organizations have adopted the Agile Development Methodology, which emphasizes iterative, incremental processes and adaptability to changing user needs. This approach has been shown to increase flexibility and development speed, making it the dominant method in modern software development (Schwaber & Sutherland, 2020; Dingsøyr et al., 2022).

On the other hand, software testing plays a crucial role in ensuring system quality and reliability. However, testing activities often require significant costs and effort, particularly in

test case design, execution, and test result analysis (Garousi & Mäntylä, 2022). This challenge is further exacerbated in agile environments due to rapidly changing requirements with each sprint iteration, requiring the testing process to be dynamically adaptable (Felderer et al., 2023).

In practical agile development settings, these challenges can be observed within a typical sprint cycle. For instance, in a two-week sprint, development teams are required to deliver multiple features based on evolving user stories. Each feature must undergo thorough testing within a limited timeframe to ensure quality before deployment. However, frequent changes in requirements often render previously designed test cases obsolete, forcing testers to continuously revise or recreate them. At the same time, testers are expected to perform test case design, regression testing, and defect analysis simultaneously. This situation creates a bottleneck, as manual testing processes demand substantial time and effort, potentially causing delays in testing activities compared to development progress. Consequently, there is an increased risk of undetected defects and compromised software quality. For example, when implementing a new feature such as an online payment module, testers must account for various scenarios, including valid and invalid transactions, edge cases, and security considerations. In a dynamic agile environment, maintaining and updating such comprehensive test cases manually becomes inefficient and prone to error.

As artificial intelligence advances, generative AI has emerged as a new approach with the potential to improve efficiency in software engineering. This technology is capable of automatically generating code, test cases, and other software artifacts. Several studies have shown that large language models can help improve developer productivity and support software testing activities, such as test scenario generation and bug analysis (Peng et al., 2023; Sobania et al., 2023; Tufano et al., 2023).

However, the use of generative AI in software testing still faces limitations, such as the need for output quality validation and challenges in integrating it into agile workflows (Felderer et al., 2023). Furthermore, previous literature studies generally focus on the use of AI in software engineering in general or on specific aspects, such as code generation or test automation, without specifically discussing implementation models for generative AI in software testing within agile environments. Several existing Systematic Literature Reviews (SLRs) also fail to provide a comprehensive map of the integration of generative AI and agile practices within the context of software testing (Garousi & Mäntylä, 2022; Sobania et al., 2023).

Based on this gap, this study aims to conduct a systematic literature review to identify, analyze, and map generative AI implementation models in software testing within the Agile Development Methodology. This research is expected to contribute to a more structured understanding of the developments, opportunities, and challenges in implementing generative AI in agile-based software testing processes.

METHOD

Research Design

This study uses the Systematic Literature Review (SLR) method to identify, evaluate, and synthesize research related to the implementation of Generative AI in software testing within an Agile development environment. The SLR method was chosen because it can provide a comprehensive understanding of research developments in a field through a systematic and replicable process.

Research Questions (RQ)

Based on the research objectives, the research questions used in this SLR are as follows:

RQ1: What are the Generative AI-based software testing frameworks or approaches used in software development?
RQ2: What are the characteristics of a Generative AI-based software testing framework in an Agile development environment?
RQ3: What are the advantages and limitations of each Generative AI-based software testing framework?
RQ4: Which Generative AI framework is most effective in improving the quality and efficiency of software testing in Agile development?

PICO Framework

To define the scope of the research and assist the literature search process, this study uses the PICO (Population, Intervention, Comparison, Outcome) framework.

Table 1. PICO Framework

Component	Description
Population (P)	Software testing in Agile-based software development
Intervention (I)	Generative AI-based software testing framework or method
Comparison (C)	Traditional testing frameworks or other AI frameworks
Outcome (O)	Improved testing efficiency, test coverage, bug detection, and software quality

The use of the PICO framework helps ensure that the literature search process focuses on research relevant to the topic of generative AI in software testing in an agile development environment.

Literature Search Strategy

The literature search process in this study was conducted systematically by utilizing several major scientific databases widely used in the field of software engineering. The databases used included IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Google Scholar. These databases were selected based on their reputation and extensive publication coverage in computer science and software engineering, enabling researchers to obtain relevant and high-quality literature. Furthermore, the use of multiple databases aimed to increase the completeness and reliability of the literature search results in this study. The publication year range used in the literature search process was 2021 to 2026, with the aim of ensuring that the analyzed studies represent the latest developments related to the application of Generative AI in software testing, particularly in the context of agile-based software development.

Search String

The search string is structured based on PICO components and keywords relevant to the research topic. The search string used is:

("Generative AI" OR "Large Language Model" OR "LLM") AND ("Software Testing" OR "Test Automation" OR "AI-based testing") AND ("Agile" OR "Agile Development" OR "DevOps") AND ("Framework" OR "Approach" OR "Method")

This search string is used in each database with syntax adjustments according to the search system of each database.

Inclusion and Exclusion Criteria

To ensure the quality and relevance of the analyzed literature, this study established inclusion and exclusion criteria in the article selection process. Articles were included in the study if they met several criteria: discussing the application of Generative AI or AI-based testing in software testing, published between 2021 and 2026, published in a journal, conference, or scientific proceeding, discussing AI frameworks or methods in software testing, and available in English.

Conversely, articles will be excluded from the study if they do not discuss software testing, do not use Artificial Intelligence or Generative AI approaches, are non-academic publications such as editorials or blogs, are not available in full text, or are duplicates from different databases. These criteria are intended to ensure that the literature used is relevant and of high quality, consistent with the research focus.

Study Selection Process

The literature selection process in this study was conducted through several stages referring to the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines to ensure transparency and reproducibility of the research. The first stage is identification, which identifies articles obtained from various databases using a predetermined search string. Next, in the screening stage, duplicate articles are removed and an initial selection is conducted based on the title and abstract to assess their suitability to the research topic. The next stage is eligibility, which involves a full-text review of articles that pass the screening stage to ensure they comply with the inclusion and exclusion criteria. Finally, in the included studies stage, articles that meet all criteria are included in the further analysis process in this study.

Data Extraction

Data from each selected article was then collected through a data extraction process to obtain information relevant to the research objectives. The extracted information included several key elements: the author(s) to identify the study source, the year of publication to assess the research's development over time, and the AI testing framework used in the study. Furthermore, the study also recorded the type of testing used, such as*unit testing*, *integration testing*, or *end-to-end testing*.

Furthermore, the data collected also includes the AI techniques used (AI technique), for example*large language models (LLM)*, *generative models*, or *reinforcement learning*This study also identifies the level of integration with agile methodology, namely how the framework is integrated into the agile-based development pipeline or process. Finally, each article is analyzed based on its primary research contribution to understand the role and innovation offered by each study in the development of a generative AI-based software testing framework. The collected information is then used as a basis for the analysis and comparison process between frameworks in the next stage.

Data Analysis

Data obtained from the literature were analyzed using qualitative synthesis and comparative analysis methods to compare various generative AI-based software testing frameworks. The analysis was conducted by grouping the frameworks based on several aspects, including the AI approach used, the type of testing supported, integration with agile development, and the framework's advantages and limitations. The results of this analysis were then used to identify the most effective framework in supporting software testing in an Agile development environment.

RESULTS AND DISCUSSION

The literature search process was conducted through several scientific databases, namely IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Google Scholar. Based on the initial search process using predetermined keywords, a total of 1,188 articles were obtained. Subsequently, the process of removing duplicate articles and screening based on titles and abstracts was carried out, significantly reducing the number of remaining articles. Articles that passed this stage then underwent a full-text review process using predetermined inclusion and exclusion criteria. After all selection stages were completed, 35 articles were obtained that met the criteria and were used in the analysis process in this study. The flow of the literature selection process can be seen in the PRISMA diagram.

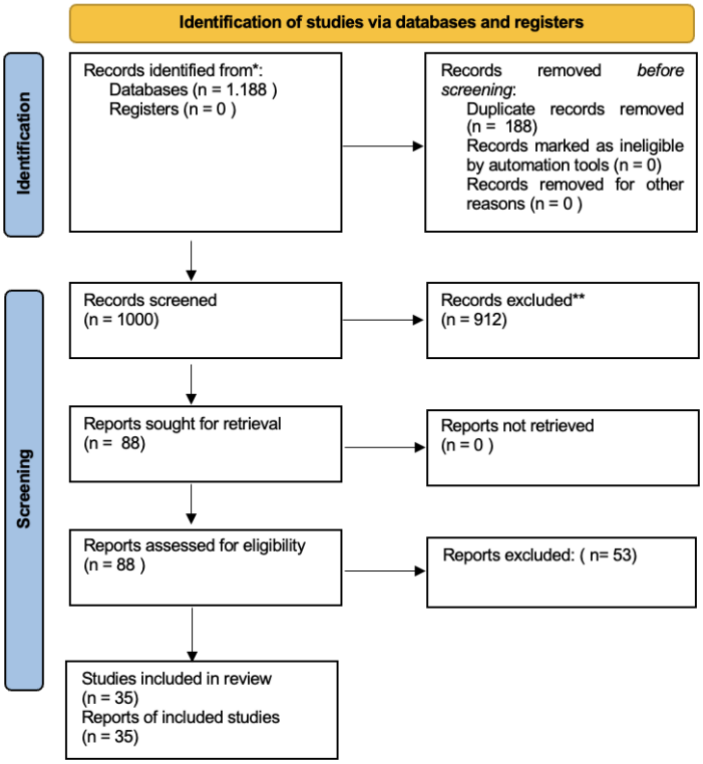

Figure 1. PRISMA Diagram

Characteristics of Selected Studies

Table 2. Study Characteristics (n=35)

No	Author	Year	Framework / Approach	Testing Type	AI Technique	Agile Integration	Contribution
1	Yahaya et al.	2025	Pairwise Test Case Generation Framework	Test case generation	AI-based optimization	Partial	SLR on pairwise test case generation
2	Nett et al.	2025	Cypress Copilot	Web testing	Generative AI / LLM	Yes	AI assistant for web testing automation
3	Ardic et al.	2025	Qualitative Testing Framework	Software testing analysis	AI-assisted analysis	Partial	Qualitative testing method mapping study
4	Ahmed et al.	2025	AI for Software Engineering Framework	Software engineering testing	Machine learning / AI	Yes	Survey of AI developments in SE

No	Author	Year	Framework / Approach	Testing Type	AI Technique	Agile Integration	Contribution
5	Pezzè et al.	2025	Software Engineering Roadmap	Software lifecycle	AI-based approaches	Yes	Roadmap software engineering 2030
6	Dong et al.	2024	Self-Collaboration Code Generation	Code generation testing	LLM / ChatGPT	Partial	AI collaboration in code generation
7	Langdon & Clark	2025	Mutation-based Testing	Mutation testing	Evolutionary algorithm	No	Mutation testing impact analysis
8	Jang & Kim	2025	NL-based Test Case Generation	Automated testing	NLP / AI	Partial	Generation of test cases from requirements
9	Who et al.	2024	A/B Testing Framework	Experiment testing	Data analytics / AI	Partial	Review metode A/B testing
10	Martensson	2025	Industrial Unit Testing Study	Unit testing	Empirical analysis	No	Industry studies on unit testing
11	Dakhel et al.	2024	LLM Test Generation Framework	Test generation	Large Language Models	Yes	Test generation using LLM
12	Baralla et al.	2024	GitHub Copilot Testing	Code testing	Generative AI	Yes	Copilot evaluation for debugging
13	Banh et al.	2025	Generative AI SE Framework	Software engineering testing	Generative AI	Yes	SE transformation by generative AI
14	Zhao et al.	2025	LLM Application Framework	Software development testing	LLM	Partial	LLM application ecosystem analysis
15	Usman et al.	2025	AI Integration Framework	Network system testing	AI / ML / LLM	Partial	AI integration in network systems
16	Sharma et al.	2024	ML for Source Code Framework	Code testing	Machine learning	Partial	ML survey for code analysis
17	Murillo et al.	2025	Quantum Software Engineering	Software system testing	AI-assisted analysis	No	Roadmap software engineering quantum
18	Mojahed et al.	2025	ODACE-RMS Testing Platform	Mobile testing	Automation framework	Yes	Automated Android testing platform
19	Durrani et al.	2025	AI Impact SE Framework	Software lifecycle	AI analytics	Partial	Analysis of the impact of AI on SE
20	Mastropaolo et al.	2025	AI Software Evolution Study	Software lifecycle testing	AI-assisted analysis	Partial	SE evolution in the AI era
21	Krebs & Mazumdar	2025	LLM Code Analysis	Code quality testing	LLM	Partial	ISO based code analysis
22	Ahlgren et al.	2025	LLM Prompt Engineering Framework	Software development testing	LLM	Yes	Prompt optimization for SE

No	Author	Year	Framework / Approach	Testing Type	AI Technique	Agile Integration	Contribution
23	Karpurapu et al.	2024	BDD Acceptance Test Framework	Acceptance testing	LLM	Yes	Automated BDD test formulation
24	Esposito et al.	2026	Generative AI Architecture Framework	Architecture testing	Generative AI	Yes	AI for software architecture
25	Torgbi Agbemabiese	2026	Agentic AI Framework	Autonomous system testing	AI Agent	Partial	Autonomous AI framework
26	Mishra & Senapati	2025	Retail Resilience Engine	System testing	AI Agent	Yes	AI framework for system reliability
27	Lano & Siala	2024	Model Driven Engineering	Software transformation testing	Model-based AI	Partial	Software language translation automation
28	Lin et al.	2025	Open-source AI SE Tools	Software tool testing	AI / LLM	Partial	Analisis tools AI open source
29	Valdivia et al.	2026	ISO/IEC 29110 AI Framework	Software development testing	Generative AI	Yes	AI integration with ISO standards
30	Abrahão et al.	2025	Human-AI SE Framework	Software lifecycle	AI collaboration	Yes	Human and AI interaction
31	Walczak et al.	2026	Automated Unit Test Generation	Unit testing	LLM	Yes	Automated unit test generation
32	Majdinasab et al.	2025	Code Dataset Detection	Code analysis testing	Machine learning	No	Model training data detection
33	Jiang et al.	2024	Self-Planning Code Generation	Code testing	LLM	Partial	LLM for code planning
34	Muttillo et al.	2025	Intelligent Modeling Assistant	Software modeling testing	LLM	Partial	AI-based modeling assistant
35	Sisomboon et al.	2026	RAG-based Test Case Generation	Automated testing	LLM + RAG	Yes	Automatic test case generation

Discussion

RQ1: Generative AI-based Software Testing Framework / Approach

Based on the results of a Systematic Literature Review of 35 studies, it was found that there are several main approaches in implementing Generative AI-based software testing, namely Large Language Model (LLM)-based, Generative AI-based, Machine Learning (ML)-based, and hybrid approaches that combine several techniques such as LLM with Retrieval-Augmented Generation (RAG) or Agentic AI.

LLM-based approaches are the most widely used in current research due to their ability to understand the context of requirements and automatically generate test cases (Dakhel et al., 2024; Walczak et al., 2026; Sisomboon et al., 2026). Furthermore, generative AI is also widely used in test automation and debugging (Banh et al., 2025; Baralla et al., 2024).

Machine learning-based approaches are still used, particularly for code analysis and pattern detection, although their use is not as widespread as LLM (Sharma et al., 2024; Majdinasab et al., 2025). Meanwhile, hybrid approaches are emerging to improve testing accuracy and effectiveness (Torgbi Agbemabiese, 2026; Mishra & Senapati, 2025).

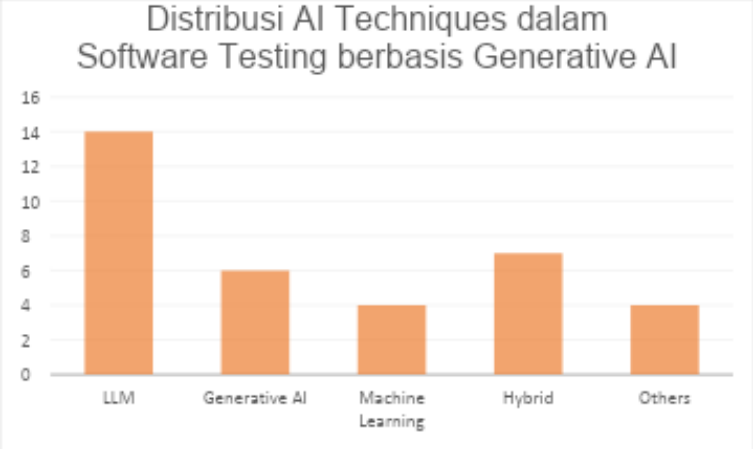

Figure 1. Distribution of AI Techniques in Generative AI-based Software Testing

Figure 1 shows that the Large Language Model (LLM)-based approach is the most dominant technique used in Generative AI-based software testing research. This dominance is evident in the significantly higher number of studies compared to other approaches, such as conventional Generative AI, Machine Learning, and hybrid approaches. The high use of LLM is due to its ability to understand the context of requirements, automatically generate test cases, and support efficient debugging and code analysis (Dakhel et al., 2024; Walczak et al., 2026; Sisomboon et al., 2026). Furthermore, LLM is also able to adapt to the dynamic needs of Agile development, so it is widely integrated into continuous testing and CI/CD pipelines (Nettur et al., 2025; Valdivia et al., 2026).

Hybrid approaches come in second, demonstrating a trend toward combining various techniques, such as LLM with RAG or Agentic AI, to improve the accuracy and relevance of test results (Torgbi Agbemabiese, 2026; Mishra & Senapati, 2025). Meanwhile, Generative AI and Machine Learning approaches remain in use, although they tend to be more limited to specific applications, such as code analysis or specific automation (Sharma et al., 2024; Banh et al., 2025). Overall, this distribution indicates a shift in research trends towards the use of language-based models as the primary solution in software testing automation, particularly in supporting efficiency, scalability, and adaptability in Agile development environments.

RQ2: Characteristics of Frameworks in Agile Development

The main characteristic of a Generative AI-based software testing framework in an Agile development environment lies in its level of integration with the iterative and continuous development process. Based on the analysis, this integration can be categorized into three levels: fully integrated, partially integrated, and non-integrated. Most studies show that frameworks support integration with Agile, either fully or partially. Fully integrated frameworks can support continuous testing, automated test case generation, and integration with CI/CD pipelines (Nettur et al., 2025; Dakhel et al., 2024; Valdivia et al., 2026). However, some studies still show partial integration, generally limited to specific stages such as test case generation or code analysis (Jang & Kim, 2025; Zhao et al., 2025). Furthermore, there are also approaches that have not been integrated with Agile because they are still conceptual or traditional (Langdon & Clark, 2025; Mårtensson, 2025).

Table 2. Characteristics of Generative AI-Based Software Testing Frameworks in Agile Development"

Framework / Approach	AI Technique	Agile Integration	Focus Testing	Main Characteristics	Study Example
Cypress Copilot	Generative AI / LLM	Yes (Full)	Web Testing	AI assistant automation testing, supports CI/CD	Nettur et al., 2025
LLM Test Generation Framework	LLM	Yes (Full)	Test Generation	Automated test case generation from requirements, Agile integration	Dakhel et al., 2024
Generative AI SE Framework	Generative AI	Yes (Full)	Software Engineering Testing	AI-based SE transformation, supporting Agile iterations	Banh et al., 2025
ISO/IEC 29110 AI Framework	Generative AI	Yes (Full)	Software Development Testing	AI integration with ISO standards, supports continuous testing	Valdivia et al., 2026
AI for Software Engineering Framework	Machine Learning / AI	Yes (Full)	Software Engineering Testing	Survey of AI developments in SE, adaptive to Agile	Ahmed et al., 2025
NL-based Test Case Generation	NLP / AI	Partial	Automated Testing	Test case generation from requirements, partial integration with Agile	Jang & Kim, 2025
Self-Collaboration Code Generation	LLM / ChatGPT	Partial	Code Generation Testing	AI collaboration in code generation, limited Agile integration	Dong et al., 2024
Agentic AI Framework	AI Agent	Partial	Autonomous System Testing	Autonomous AI framework, supporting multiple Agile iterations	Torgbi Agbemabiese, 2026
Mutation-based Testing	Evolutionary Algorithm	No	Mutation Testing	Mutation testing impact analysis, not related to Agile	Langdon & Clark, 2025
Industrial Unit Testing Study	Empirical Analysis	No	Unit Testing	Industry study on unit testing, non-Agile	Martensson, 2025

Based on Table 2, it can be seen that the level of integration of Generative AI-based software testing frameworks with Agile development varies, ranging from full to partial integration, and some are still not integrated at all. Fully integrated frameworks (Yes), such as Cypress Copilot (Nettur et al., 2025), LLM Test Generation Framework (Dakhel et al., 2024), and Generative AI SE Framework (Banh et al., 2025), are capable of supporting end-to-end testing automation, integration into CI/CD pipelines, and the continuous testing process that is at the heart of Agile development. This approach demonstrates that the use of Generative AI not only improves the efficiency of test case generation but also ensures

testing proceeds iteratively according to Agile sprints (Valdivia et al., 2026; Abrahão et al., 2025).

Meanwhile, some frameworks only have partial integration, for example NL-based Test Case Generation (Jang & Kim, 2025), Self-Collaboration Code Generation (Dong et al., 2024), and the Agentic AI Framework (Torgbi Agbemabiese, 2026). These frameworks support some aspects of Agile, such as test case generation or AI collaboration in coding, but are not yet fully integrated throughout the development cycle, so their use is still limited to certain stages. This shows that although Generative AI has great potential, full implementation in an Agile environment requires infrastructure and workflow adjustments (Ahmed et al., 2025; Zhao et al., 2025). On the other hand, there are several frameworks that are not integrated with Agile (No), such as Mutation-based Testing (Langdon & Clark, 2025) and Industrial Unit Testing Study (Mårtensson, 2025). These frameworks are generally conceptual or empirical in nature and focus more on traditional testing methods, thus not supporting the rapid iteration and continuous testing that are characteristic of Agile. Overall, this distribution of characteristics indicates a growing trend of Generative AI integration into Agile practices, particularly for LLM-based frameworks and pure Generative AI. Hybrid frameworks are also emerging to improve testing flexibility and accuracy, despite their higher implementation complexity (Torgbi Agbemabiese, 2026; Mishra & Senapati, 2025). This confirms that adopting Generative AI in Agile development not only accelerates the testing process but also requires thoughtful workflow design and sprint management for maximum effectiveness.

RQ3: Strengths and Limitations of the Framework

The analysis shows that each approach has distinct advantages and limitations. LLM-based frameworks and Generative AI offer a high degree of automation, particularly in test case generation and debugging (Walczak et al., 2026; Baralla et al., 2024). Furthermore, these approaches are adaptable to dynamic requirements changes in Agile development (Banh et al., 2025). However, the main limitation of this approach is its dependence on the quality of input (prompt) and the potential for hallucinations which can produce less accurate output (Krebs & Mazumdar, 2025; Ahlgren et al., 2025). On the other hand, machine learning-based approaches tend to be more stable and structured in analysis, but less flexible in dealing with rapid changes (Sharma et al., 2024). Hybrid approaches offer solutions by combining the advantages of various techniques, although implementation complexity poses challenges (Torgbi Agbemabiese, 2026).

Table 3. Summary of Strengths and Limitations of the Framework

AI Technique	Advantages	Limitations	Study Example
LLM	High automation, adaptive to Agile	Sensitive to prompt quality, risk of hallucination	Dakhel et al., 2024; Walczak et al., 2026; Sisomboon et al., 2026
Generative AI	Efficient, speeds up testing	Output is not always accurate, requires a mature workflow	Banh et al., 2025; Baralla et al., 2024
Machine Learning	Stable, suitable for code analysis	Less flexible, limited Agile integration	Sharma et al., 2024; Majdinasab et al., 2025
Hybrid / Multi-AI	High accuracy, adaptive	High implementation complexity	Torgbi Lifeless, 2026; Mishra & Senapati,

Based on Table 3, it can be seen that each type of AI in the software testing framework has its own advantages and limitations.

LLM-based frameworks demonstrate dominance in automating test case generation and debugging processes, and are highly adaptable to changing requirements in Agile

development (Dakhel et al., 2024; Walczak et al., 2026; Sisomboon et al., 2026). This makes LLM a prime choice for teams seeking to improve testing efficiency and flexibility. However, reliance on prompt quality and the risk of hallucinations remain key limitations, which can impact the accuracy of testing results (Krebs & Mazumdar, 2025; Ahlgren et al., 2025). Pure Generative AI approaches also provide significant efficiencies in testing automation and support rapid iteration in Agile development (Banh et al., 2025; Baralla et al., 2024). However, the resulting output is not always consistent, and its implementation requires a well-developed workflow for optimal Agile integration.

Machine learning-based frameworks are more stable and suitable for code analysis or specific pattern detection (Sharma et al., 2024; Majdinasab et al., 2025). However, their flexibility is limited when dealing with rapidly changing requirements, so integration with Agile processes is often partial. Meanwhile, hybrid or multi-AI frameworks like the Agentic AI Framework combine the advantages of various AI techniques, improving the accuracy and relevance of testing in an Agile context (Torgbi Agbemabiese, 2026; Mishra & Senapati, 2025). These advantages make hybrid frameworks a potential solution for teams looking to maximize testing quality. However, implementation complexity poses a major challenge, both technically and in terms of workflow management. Overall, this distribution of advantages and limitations underscores the trend toward using LLM and Generative AI as the primary frameworks for automated software testing in Agile development, while ML and hybrid frameworks are used for specific cases requiring stability or a combination of AI's advantages. This also emphasizes the importance of adapting workflows, data quality, and engineering prompts for optimal framework implementation (Nettur et al., 2025; Valdivia et al., 2026).

RQ4: The Most Effective Framework

Based on the results of literature analysis, Large Language Model (LLM) and Generative AI-based frameworks are the most effective approaches in improving the quality and efficiency of software testing in Agile development. This dominance is demonstrated not only by their high frequency of use in various studies but also by their significant contribution in supporting comprehensive test automation, from the planning stage to test execution (Dakhel et al., 2024; Sisomboon et al., 2026; Walczak et al., 2026). More specifically, this framework can increase efficiency by automating test case generation, which previously required significant time and resources when done manually. Furthermore, LLM's ability to understand the context of requirements enables the creation of more relevant and comprehensive test scenarios. During the execution phase, Generative AI also plays a role in accelerating the debugging and error analysis process, allowing for faster improvement cycles in each Agile sprint (Nettur et al., 2025; Valdivia et al., 2026). Another advantage is its ability to support continuous testing through integration with Agile pipelines like CI/CD. This integration allows testing to be performed automatically whenever code changes occur, continuously improving software quality. This aligns with Agile principles, which emphasize rapid iteration and continuous feedback. Beyond efficiency, LLM and Generative AI-based frameworks also demonstrate a high degree of adaptability to changing requirements, a key characteristic of Agile development. This model can dynamically adjust test cases based on changing user stories or system requirements, thereby reducing the need for rework in the testing process.

However, the effective implementation of this framework is not without several challenges. One key factor is the dependence on the quality of the prompts and input data, which directly impacts the accuracy of the resulting output. Mistakes in prompt design can lead to irrelevant or even misleading test results. Furthermore, there is a risk of hallucination in the LLM model, which can produce output that does not reflect the actual system

conditions (Ahlgren et al., 2025). Other factors influencing effectiveness are infrastructure readiness and system integration. Generative AI implementation in an Agile environment requires adequate technological support, including an integrated CI/CD pipeline and team capability in managing AI-based workflows. Without such support, the framework's potential benefits cannot be fully realized. Overall, while LLM and Generative AI-based frameworks have demonstrated high effectiveness in improving the quality and efficiency of software testing, their success depends heavily on implementation strategy, input quality, and organizational readiness to adopt AI technology. Therefore, this approach requires not only technical innovation but also process and management adjustments within an Agile development environment.

CONCLUSION

This study conducted a Systematic Literature Review (SLR) of 35 articles discussing the application of Generative AI in software testing within an Agile development environment. The study results show that Artificial Intelligence technology, particularly Large Language Models (LLM) and various generative models, have been utilized to support the automation of software testing processes, such as automatic test case generation, code analysis, and behavior-driven development-based testing. Furthermore, several studies have developed frameworks and AI assistants that can be integrated into the software development process to improve team efficiency and productivity.

The findings of this study indicate that the application of Generative AI has significant potential to increase the speed and efficiency of the software testing process, especially in Agile environments that demand rapid and adaptive development cycles. However, the effectiveness of this technology still depends on model quality, data availability, and human validation processes. Therefore, a hybrid approach between AI and testers is still necessary to maintain system quality and reliability.

Practically, the results of this study can serve as a reference for software developers and software testers in implementing Generative AI in the testing process. Generative AI can be utilized to accelerate test case creation in each sprint, assist with bug identification, and support regression testing automation. Nevertheless, practitioners are advised to continue verifying AI-generated output and gradually integrating this technology into existing Agile workflows. Furthermore, organizations need to establish evaluation and quality control standards to ensure that the use of AI does not compromise software reliability and security.

Future Work

Based on the results of the Systematic Literature Review that has been conducted, several opportunities for further research that can be developed are as follows:

1. Development of a standardized implementation model

   Future research could develop a more structured and specific model or framework for implementing generative AI in software testing for Agile Development environments. This is important because most existing studies are still conceptual or limited to specific case studies.
2. Empirical evaluation in real projects (real-world validation)

   Most research is still conducted in experimental settings. Therefore, future research needs to conduct empirical validation through direct implementation on agile software projects to measure the effectiveness of generative AI on software quality, testing speed, and cost efficiency.
3. Improving the quality and reliability of AI output

Further research is needed to improve the accuracy and consistency of generative AI results, particularly in test case generation and bug detection. This includes developing automated evaluation methods to minimize reliance on manual validation.

4. Integrating generative AI with DevOps and CI/CD tools

   Further research could explore the integration of generative AI into DevOps pipelines, such as continuous integration and continuous testing, to support end-to-end test automation in agile environments.

5. A study on the impact on the role and competence of testers

   The use of generative AI has the potential to transform the role of software testers. Therefore, further research can examine how human competencies, skills, and roles in the software testing process will change in the AI era.

6. Analysis of security and ethical aspects

   Further research should also address the security risks, biases, and potential vulnerabilities created by the use of generative AI in software testing, particularly in critical systems.

7. Development of hybrid techniques (Human-AI Collaboration)

   Future research can develop a collaborative approach between humans and AI in software testing to maximize the advantages of both, resulting in a more optimal testing process

REFERENCE

Ahmed, I., Aleti, A., Cai, H., Chatzigeorgiou, A., He, P., Hu, X., Pezzè, M., Poshyvanyk, D., & Xia, X. (2025). Artificial intelligence for software engineering: The journey so far and the road ahead. *Communications of the ACM, 34*(5), 119. https://doi.org/10.1145/3719006

Ahlgren, T. L., Sunde, H. F., Kemell, K. K., & Nguyen-Duc, A. (2025). Assisting early-stage software startups with LLMs. *Information and Software Technology, 187*, 107832. https://doi.org/10.1016/j.infsof.2025.107832

Ardic, B., Brandt, C., Khatami, A., Swillus, M., & Zaidman, A. (2025). The qualitative factor in software testing: A systematic mapping study of qualitative methods. *Journal of Systems and Software, 227*, 112447. https://doi.org/10.1016/j.jss.2025.112447

Banh, L., Holldack, F., & Strobel, G. (2025). Copiloting the future: How generative AI transforms software engineering. *Information and Software Technology, 183*, 107751. https://doi.org/10.1016/j.infsof.2025.107751

Baralla, G., Ibba, G., & Tonelli, R. (2024). Assessing GitHub Copilot in Solidity development: Capabilities, testing, and bug fixing. *IEEE Access, 12*, 164389–164411. https://doi.org/10.1109/ACCESS.2024.3486365

Dakhel, A. M., Nikanjam, A., Majdinasab, V., Khomh, F., & Desmarais, M. C. (2024). Effective test generation using pre-trained large language models and mutation testing. *Information and Software Technology, 171*, 107468. https://doi.org/10.1016/j.infsof.2024.107468

Dong, Y., Jiang, X., Jin, Z., & Li, G. (2024). Self-collaboration code generation via ChatGPT. *ACM Transactions on Software Engineering, 33*(7), 189. https://doi.org/10.1145/3672459

Durrani, U. K., Akpinar, M., Bektas, H., & Saleh, M. (2025). Impact of artificial intelligence on software engineering phases and activities. *IEEE Access, 13*, 95535–95547. https://doi.org/10.1109/ACCESS.2025.3574462

Esposito, M., Li, X., Moreschini, S., Ahmad, N., Cerny, T., Vaidhyanathan, K., Lenarduzzi, V., & Taibi, D. (2026). Generative AI for software architecture. *Journal of Systems and Software, 231*, 112607. https://doi.org/10.1016/j.jss.2025.112607

Jang, W., & Kim, R. Y. C. (2025). Automatic test case generation mechanism with natural language-based Korean requirement specifications. *IEEE Access, 13*, 177305–177317. https://doi.org/10.1109/ACCESS.2025.3620431

Karpurapu, S., Myneni, S., Nettur, U., Gajja, L. S., Burke, D., Stiehm, T., & Payne, J. (2024). Comprehensive evaluation and insights into the use of LLMs in BDD acceptance test formulation. *IEEE Access, 12*, 58715–58721. https://doi.org/10.1109/ACCESS.2024.3391815

Krebs, R., & Mazumdar, S. (2025). Analyzing LLM-generated code according to ISO/IEC 5055:2021 categories. *IEEE Access, 13*, 202482–202499. https://doi.org/10.1109/ACCESS.2025.3637569

Langdon, W. B., & Clark, D. (2025). Deep imperative mutations have less impact. *Automated Software Engineering, 32*(1), 6. https://doi.org/10.1007/s10515-024-00475-4

Mårtensson, T. (2025). So much more than test cases – An industrial study on testing of software units and components. *Journal of Systems and Software, 228*, 112479. https://doi.org/10.1016/j.jss.2025.112479

Mastropaolo, A., Escobar-Velásquez, C., & Linares-Vásquez, M. (2025). From triumph to uncertainty: The journey of software engineering in the AI era. *Communications of the ACM, 34*(5), 131. https://doi.org/10.1145/3709360

Mojahed, S., Drouin, R., & Sboui, L. (2025). ODACE-RMS: A remote web-based platform for automated multi-device Android testing and certification. *IEEE Access, 13*, 99863–99878. https://doi.org/10.1109/ACCESS.2025.3576823

Murillo, J. M., Garcia-Alonso, J., Moguel, E., Barzen, J., Leymann, F., Ali, S., et al. (2025). Quantum software engineering: Roadmap and challenges ahead. *Communications of the ACM, 34*(5), 154. https://doi.org/10.1145/3712002

Nettur, S. B., Karpurapu, S., Nettur, U., & Gajja, L. S. (2025). Cypress Copilot: Development of an AI assistant for boosting productivity and transforming web application testing. *IEEE Access, 13*, 3215–3229. https://doi.org/10.1109/ACCESS.2024.3521407

Pezzè, M., Abrahão, S., Penzenstadler, B., Poshyvanyk, D., Roychoudhury, A., & Yue, T. (2025). A 2030 roadmap for software engineering. *Communications of the ACM, 34*(5), 118. https://doi.org/10.1145/3731559

Quin, F., Weyns, D., Galster, M., & Silva, C. C. (2024). A/B testing: A systematic literature review. *Journal of Systems and Software, 211*, 112011. https://doi.org/10.1016/j.jss.2024.112011

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H., & Sarro, F. (2024). A survey on machine learning techniques applied to source code. *Journal of Systems and Software, 209*, 111934. https://doi.org/10.1016/j.jss.2023.111934

Sisomboon, W., Kaewyotha, J., & Songpan, W. (2026). Automated software test case generation using ensemble LLM with retrieval-augmented generation. *IEEE Access*. https://doi.org/10.1109/ACCESS.2026.3667925

Usman , Y. , Oladipupo , H. , During , A. D. , Akl , R. , & Chataut , R. (2025). AI, ML, and LLM integration in 5G/6G networks: A comprehensive survey. *IEEE Access, 13*, 168914–168950. https://doi.org/10.1109/ACCESS.2025.3608736

Yahaya, M. S., Hashim, A. S. B., Oluwagbemiga Balogun, A., Muazu, A., Usman, F. S., Aliyu, D. A., & Muhammad, A. U. (2025). Exploration and exploitation mechanism in pairwise test case generation: A systematic literature review. *IEEE Access, 13*, 82342–82377. https://doi.org/10.1109/ACCESS.2025.3566163

Walczak, J., Tomalak, P., & Laskowski, A. (2026). Impact of code context and prompting strategies on automated unit test generation with modern large language models. *Journal of Systems and Software, 237*, 112834. https://doi.org/10.1016/j.jss.2026.112834

Zhao, Y., Hou, X., Wang, S., & Wang, H. (2025). LLM App store analysis: A vision and roadmap. *Communications of the ACM, 34*(5), 125. https://doi.org/10.1145/3708530