International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 6, Nov-Dec-2026, ISSN (Online): 2395-566X

Impact of Generative AI PN The Software Testing Industry: A Study of Benefits, Challenges, And Future Prospects

Niranjan Sandip Jagtap[1], Professor Dr. Deepak Singh[2],
Msc Computer Science,
Suryadatta College Of Management Information Research & Technology, Pune, India

Abstract- The software testing sector is seeing a growing trend of the use of Generative Artificial Intelligence (GenAI) to enhance the efficiency and effectiveness of software testing activities. Creating test cases, generating test data, identification and fixing defects and keeping testing documentation are some of the traditional software testing tasks which take a lot of time and human efforts. Generative AI can help testers to create test cases, propose test scenarios, study application behavior, and aid in defect detection and reporting. This study aims to explore the effect of Generative AI on the software testing industry through the analysis of the key advantages, challenges, and future directions. The significant factors which are taken into consideration for the research are testing efficiency, test coverage, time consumption, accuracy, cost reduction and role of human testers. It also explores the potential of using GenAI tools for various testing tasks, such as test case generation, test script development, test data generation, test regression and defect analysis. The research highlights the potential challenges in AI-powered testing, including the accuracy and completeness of the test cases generated by AI, security and privacy issues, reliance on AI-generated outputs, the absence of explainability, and the requirement for human validation. This research is expected to equip software professionals with an applied knowledge of the impact of Generative AI on software testing and on how AI can be used to augment current software testing processes. The results may provide insights for software testing experts and organizations to consider when evaluating the potential of Generative AI and deciding on its implementation while continuing to value the role of humans and quality assurance.

Keywords: Generative AI, Software Testing, Test Automation, Test Case Generation, Defect Detection, Testing Efficiency, AI in Quality Assurance, Software Quality, Human-AI Collaboration

I. INTRODUCTION

A major part of the Software Development Life Cycle (SDLC) is software testing. Creating test cases, developing test data, writing and maintaining test automation scripts, executing regression suites and reporting bugs require ongoing human resources and the consequences of inadequate testing are significant. Under Agile and DevOps practices, the release cycles are getting shorter, and test teams are expected to provide wider coverage in shorter time.

The pressure can be tackled with a new solution: Generative AI based on large language models (LLMs), which are derived from the Transformer architecture [1]. Large corpora, like Codex [2] and GPT-3 [3] that contain code and text can be used to train models that can read requirements, user stories, and source code, and generate test cases, code for implementing tests, synthetic test data, and defect summaries from plain-

language prompts [4]. AI tools like ChatGPT, GitHub Copilot, Gemini, and Claude, which are able to complete tasks such as generating test code, have transitioned from experimentation to daily QA use. AI-powered tools that are built into commercial testing platforms have done the same.

Even with this adoption, there is limited evidence of the actual impact of GenAI on testing practice. The ability to rapidize and decrease costs is an advantage offered by vendor material, whereas independent research identifies issues of correctness, completeness, and trustworthiness in AI-generated test artifacts [4] [5]. This paper tackles that blindspot by presenting a structured and narrative review of the available empirical evidence on GenAI in software testing and conducting a comparative analysis.

1. Research Problem

While evidence and governance are still being developed to effectively use GenAI in testing, software organisations are increasingly turning to such tools. This study is motivated by three issues.

The more obvious advantages like quicker test production and increased test volume are easy to quantify and are widely advertised. Secondly, the expenses aren't easily visible. A passing suite can lull the developer into a false sense of security because a generated test can compile and run, but assert the wrong behavior, or it may fail to capture the

© 2026 IJSRET

important edge cases, or it may simply be a copy of the current implementation. Third, requirements, source code, and even production-like data may be sent to an external AI service without a clear policy on privacy, IP ownership, and/or human review.

The profession must have a more comprehensive perspective of benefits, challenges, and future direction than the productivity based perspective.

2. Research Objectives

This study pursues the following objectives:

- O1: To examine how Generative AI is used across software testing activities, including test case generation, test script creation, test data generation, regression testing, and defect analysis.
- O2: To evaluate the benefits of GenAI adoption in terms of testing efficiency, test coverage, time consumption, accuracy, and cost reduction.
- O3: To identify and analyze the challenges of GenAI in testing, including inaccurate or incomplete test cases, security and privacy concerns, over-dependence, and lack of explainability.
- O4: To understand the changing role of human testers and the importance of human validation in AI-assisted testing.
- O5: To propose practical recommendations for organizations to integrate GenAI into existing testing processes responsibly.

II. LITERATURE REVIEW

1. Evolution Of Test Automation

There are several phases in the evolution of test automation. Early methods were based on manually scripted tests and record and playback tools. Covered test generation techniques based on search, like EvoSuite [6]

brought automated unit test generation based on coverage criteria. Later, machine learning methods were used for test prioritisation, detection of flaky tests, and visual testing. The current stage is based on LLMs that can create tests from the natural language context and code without being custom-made for the project [4].

2. Llms For Test Generation

Recent studies directly assess LLMs for the task of generating units. Yuan et al. explored ChatGPT and discovered that, while the tests were skilful, they had often compilation or assertion errors and became better when the model was given feedback and repair loops [5]. Schafer et al. presented TestPilot, an LLM that is guided by function signatures and documentation, and uses execution feedback to improve tests and achieve coverage beyond a state-of-the-art feedback-directed baseline for JavaScript packages [7]. To overcome coverage plateaus, Lemieux et al. used search-based testing in conjunction with an LLM (CodaMosa) [8]. Kang et al. demonstrated that LLM can generate bugs that are reported as executable tests [9]. The survey of Wang et al. characterizes this landscape and pinpoints test generation, bug analysis and program repair as the primary applications [4]. By the end of this session, students will be able to perform

GenAI in Test Data, GUI Testing and Defect Analysis.

In addition to unit tests, LLM applications have been seen creating context-aware text inputs for mobile GUI testing [10], creating synthetic and edge-case test data, creating test documentation, and summarizing and classifuing defect reports. Uses decrease the effort required but the reliability is dependent on the quality of the prompt and the context provided.

3. Limitations: Test Adequacy, The Oracle Problem And Security

Poor test suites may allow bad code to get in. Many of the solutions that seemed correct were found when Liu et al. added the tests to a popular code-generation benchmark [11]. A more fundamental challenge is the test oracle problem, in which a decision is made about what the expected result is [12]. If an LLM generates both code and tests based on a wrong understanding, the tests can then be wrong. AI code evaluations also yield security issues: In security-related scenarios, approximately 40 percent of Copilot-generated programs have been reported to be vulnerable by Pearce, et al. [13]. The results are not deterministic between different runs of the same prompt [14] further complicating reproducibility.

3. Human-Ai Collaboration And Automation Bias

Automation bias research indicates that users are more likely to trust well-structured and believable automated information, but to pay less attention to automated information that is less well structured or less convincing [15]. There are controlled studies that show a definite speed advantage for AI coding assistants; for example, Peng et al. found users could finish a programming task 55.8 percent quicker in their experiment [16] (but that's for development, not testing). When it comes to testing, the products created by polished AI might not be reviewed as thoroughly as they should be, and this will still be conducted by a human.

5 Governance, Privacy, And Regulation

Sharing sensitive data, such as proprietary requirements, source code, or customer-like information, with third-party AI services introduces potential data-protection and confidentiality concerns. Formal policy and

© 2026 IJSRET

review are important aspects of putt-into-practice, influenced by policy frameworks provided by the EU AI Act [17] and the Digital Personal Data Protection Act, 2023 [18].

III. METHODOLOGY

The study is structured narrative review which is comparative analysis of secondary data. It is based on the review guidelines of Kitchenham and Charters [19] and the narrative synthesis guidelines of Popay et al. [20]. No primary data was collected and no survey/ interview results are reported.

Review questions. (RQ1) What are the testing activities for which there is empirical evidence of GenAI benefit? (RQ2) What is the evidence of the accuracy, extent and credibility of AI generated test artifacts? (RQ3) What are the human, security and governance issues raised? (RQ4) What practices are supported by the evidence for responsible adoption?

Sources and search. Combining generative AI, large language model, software testing, test generation, test automation, and AI-generated code in various combinations, the IEEE Xplore, ACM Digital Library, arXiv, and Google Scholar were used to find peer-reviewed papers, surveys and primary regulatory documents. Additional studies were identified by conducting broad reviews of LLM's in software engineering [21].

Inclusion criteria. (i) published since 2021, and focused on LLMs or GenAI for testing, or the trustworthiness of AI-generated code for testing purposes; (ii) an empirical evaluation, survey or review; or (iii) written in English. No vendor marketing that didn't include a method and opinion pieces without evidence were included.

Selection and scope. Selection was purposive: Peer-reviewed, well-cited and replicable work was given priority. This paper is based on the reference list. The review is therefore not a systematic review, and the conclusions are limited as such.

Analysis. The testing task, technique, reported metric, result, and stated limitation were gathered from each of the sources and mapped to the testing dimensions in the objectives. The authors used the following criteria to make judgments on the strength of evidence for each task: Study design (controlled evaluation versus preliminary) and breadth (number of subjects and languages) and independence. The ratings are not statistical rates but rather judgments.

Analytical propositions. The evidence was used to evaluate five propositions:

- P1: GenAI improves the speed and coverage of routine test creation tasks.
- P2: AI-generated tests are stronger on surface qualities (compilability, coverage) than on structural qualities (oracle correctness, fault detection, security).
- P3: GenAI output is non-deterministic and context-dependent, which limits reproducibility without controls. P4: Human review remains necessary because of the oracle problem and automation bias.
- P5: Privacy, intellectual property, and regulatory exposure grow without formal governance.

IV. EVIDENCE ANALYSIS

1. Efficiency And Coverage By Testing Task

There is strong evidence that it is beneficial for unit test generation and moderate evidence otherwise for other activities (Table 1). The npm packages studied had a median statement coverage of 70.2 percent and branch coverage of 52.2 percent with TestPilot.TestPilot achieved a median statement coverage of 70.2 percent and branch coverage of 52.2 percent on the npm packages studied [7]. We reproduced approximately 1/3 of the Defects4J bugs in our study (reported as 33.5 percent) [9]. These are both benchmark results and may not apply to industrial code.

Table 1: Evidence of GenAI Benefit by Testing Task (authors' appraisal)

Testing Task	Representative Sources	Reported Outcome	Evidence Strength
Unit test generation	[5][7][8]	Competitive coverage achievable; raw output often needs repair to compile and pass	Moderate
Bug reproduction from reports	[9]	About one third of studied bugs reproduced as executable tests	Moderate
GUI and input generation	[10]	Context-aware inputs improved mobile GUI testing over baselines	Limited
Test data generation	[4]	Identified as an application; few controlled comparisons	Limited
Documentation and defect reporting	[4]	Identified as an application; evidence mostly descriptive	Limited
Regression test maintenance	[4]	Emerging area; few rigorous evaluations	Limited

© 2026 IJSRET

2. Surface Versus Structural Quality

Published evidence categorises qualities that are easy to see from those of most importance for fault detection (Table 2). Though generated tests may achieve good test coverage, the achieved coverage does not necessarily indicate correctness of assertions or that faults would be detected.

Table 2: Surface vs. Structural Quality of AI-Generated Tests

Quality Dimension	Category	Published Evidence	Source
Compilability and passing	Surface	Raw tests often fail to compile or pass; repair loops improve results	[5][7]
Code coverage	Surface	Comparable to or better than baseline tools in studied settings	[7][8]
Assertion (oracle) correctness	Structural	Deciding correct output is a fundamental unsolved problem	[12]
Test adequacy	Structural	Weak tests let incorrect code appear correct	[11]
Security	Structural	About 40% of Copilot programs in security scenarios were vulnerable	[13]
Reproducibility	Structural	Outputs vary across repeated runs of the same prompt	[14]

3. Over-Dependence And Human Factors

Fluent, well-formulated AI output is more likely to attract less scrutiny, as predicted by automation bias [15]. This effect is not measured in any study mentioned here, but that does not mean that it is not a reasonable risk, but rather a risk nonetheless. Further, since speed gains for developers [16], it appears that the review time is a limiting constraint and not the writing time.

4 Governance And Privacy

The EU AI Act [17] imposes obligations based on the level of risk of the AI system, while India's Digital Personal Data Protection Act, 2023 [18] regulates the processing of personal data. Thus, teams that copy and paste production information with personal information into external AI applications could run the risk of generating compliance liabilities. These sources were regulatory and conceptual in nature, and there was no evidence of a large-scale measurement of governance practice in testing teams.

6. Proposition Evaluation

Table 3 summarizes how well the evidence supports each proposition. The assessments are qualitative and do not involve statistical significance tests.

P	Proposition	Evidence Base	Assessment
P1	Speed and coverage gains in routine test creation	[5][7][8][9]	Supported (moderate evidence)
P2	Surface quality stronger than structural quality	[5][7][11][12][13]	Supported (moderate evidence)
P3	Non-determinism limits reproducibility	[14]	Supported, with limited direct evidence on tests
P4	Human review remains necessary	[11][12][15]	Supported by theory and indirect evidence; direct testing studies are few
P5	Exposure grows without governance	[17][18]	Partially supported; regulatory analysis, little incident data

Table 3: Evaluation of Analytical Propositions

© 2026 IJSRET

V. FINDINGS AND DISCUSSION

1. The Efficiency-Reliability Trade-Off

The evidence is clear: teams are better able to monitor speed and volume when it comes to creating test artifacts via GenAI. More challenging to measure and in which independent studies find shortcomings are correctness, depth and trustworthiness [5][7][11][12]. This imbalance can cause organisations to think that AI-driven testing is more important than it really is. Table 4 illustrates the contrast between traditional and AI-driven testing, highlighting its selected dimensions.

Table 4: Traditional vs. AI-Assisted Software Testing (Selected Dimensions**)**

Testing Dimension	Traditional Testing	AI-Assisted Testing	Net Assessment
Test case creation speed	Manual, time-consuming	Rapid generation from requirements or code	Clear AI advantage
Test data generation	Manual or script-based	Synthetic, diverse, edge-case data	AI advantage, with privacy checks
Oracle correctness	Human judgment	May mirror the implementation or contain errors	Human advantage
Regression maintenance	Costly script upkeep	Assisted script repair and updates	Promising, needs validation
Exploratory and domain testing	Tester intuition and experience	Limited contextual understanding	Human advantage
Explainability	Traceable to tester reasoning	Opaque model behavior	Human advantage
Data security	Controlled in-house	Risk of exposure to external services	Depends on governance

2. Automation Bias And Human Validation Requirements.

AI-generated tests tend to be syntactically correct, well-formatted, and well-named. The automation bias framework [15] states that such polishing can decrease reviewers' scrutiny of the paper, in which case weak assertions or missing scenarios might be overlooked. Small faults are inserted into assess the effectiveness of generated tests of the test suite (with respect to the faults), called mutation testing [22]. Requirements for review checklists and fault detection measures provide additional protection, beyond that which is provided through individual vigilance.

3. The Role Of The Tester Is Evolving.

GenAI transforms testers from writing all artifacts to prompting, reviewing, and curating AI's output; and from low-value tasks to more valuable ones like risk analysis, exploratory testing, and test strategy. This is good for testers with a problem-solving attitude who are knowledgeable about the test domain, critical evaluators, and tooling literacy.

4. Privacy, Security, And Governance

Organizations should identify what can be shared with AI services, opt for enterprise-wide protocols or deployments of their own, mask and/or synthesize production data if disclosing it is not permitted, and document the use of AI in connection with their test assets [17] [18].

5. Recommendations

Treat AI-generated tests as drafts that always require human review.

- Measure quality through fault detection, such as mutation score, and not only test count or coverage.
- Establish a written GenAI usage policy covering data handling, approved tools, and review responsibilities. Start with low-risk tasks such as documentation, boilerplate scripts, and test data, then expand gradually.
- Train testers in prompt design, evaluation of AI output, and AI risk awareness.
- Add automated checks, including security scanning, to the CI/CD pipeline for AI-assisted artifacts.

6. Limitations

This is a purposive narrative review, not a systematic review - relevant studies may be omitted. The most frequently referenced evaluations are based on benchmarks, specific languages (such as Python, Java, JavaScript), and specific versions of the models; the landscape of tools is rapidly evolving. The authors' judgements about the strength of the evidence. Primary data were not gathered, with claims

© 2026 IJSRET

relating to practitioner attitudes and/or organizational practice based on indirect evidence.

VI. CONCLUSION

This paper summarizes the effect of Generative AI on software testing in the areas of efficiency, software quality, human role, and governance. The findings that have been published indicate that there are clear benefits when using GenAI for generating tests, reproducing bugs and performing routine tasks like test data generation or documentation. There are also some big hurdles to these benefits: tests that are suboptimal, the oracle problem, inadequate test coverage, vulnerabilities, and other non-deterministic outputs, privacy and regulatory issues, and the danger of becoming dependent on tests. Four of the five propositions tested in the analysis were confirmed, and one partially. The key takeaway is GenAI should be used as a collaborative tool, not a replacement for tester judgement, under human oversight and governance. The combination of AI's rapid speeds and metrics for fault detection, review practices, and a defined policy is likely to be the best approach for enhancing both productivity and software quality.

1. Future Scope

Future research opportunities include: (i) Empirical studies with practitioners to gain insight into adoption, perceived utility and governance practices in testing. This was not possible in this review. (ii) Standardised metrics to measure how well AI-generated tests find faults. (iii) Studies into automation bias in reviewing AI-generated tests. (iv) Studies into autonomous testing systems, including their ability to automatically plan, perform and recover tests. (v) Improving explainability, traceability and re-producibility in AI generated testing artefacts. (vi) Solutions for preserving privacy and in situ testing. Studies in safety-critical and regulated domains (e.g., healthcare and finance).

REFERENCES

1. A. Vaswani et al., Attention is all you need, Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.
2. M. Chen et al., Evaluating large language models trained on code, arXiv:2107.03374, 2021.
3. T. Brown et al., Language models are few-shot learners, Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.
4. J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, Software testing with large language models: Survey, landscape, and vision, IEEE Transactions on Software Engineering, vol. 50, no. 4, pp.911-936, 2024.
5. Z. Yuan et al., No more manual tests? Evaluating and improving ChatGPT for unit test generation, arXiv:2305.04207, 2023.
6. G. Fraser and A. Arcuri, EvoSuite: Automatic test suite generation for object-oriented software, Proc. ESEC/FSE, 2011, pp. 416-419.
7. M. Schafer, S. Nadi, A. Eghbali, and F. Tip, An empirical evaluation of using large language models for automated unit test generation, IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85-105, 2024.
8. C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, CodaMosa: Escaping coverage plateaus in test generation with pre-trained large language models, Proc. ICSE, 2023, pp. 919-931.
9. S. Kang, J. Yoon, and S. Yoo, Large language models are few-shot testers: Exploring LLM-based general bug reproduction, Proc. ICSE, 2023, pp. 2312-2323.
10. Z. Liu et al., Fill in the blank: Context-aware automated text input generation for mobile GUI testing, Proc. ICSE, 2023,pp. 1355-1367.
11. J. Liu, C. S. Xia, Y. Wang, and L. Zhang, Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation, Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
12. E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, The oracle problem in software testing: A survey, IEEE Transactions on Software Engineering, vol. 41, no. 5, pp. 507-525, 2015.
13. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions, Proc. IEEE Symposium on Security and Privacy, 2022, pp. 754-768.
14. S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, LLM is like a box of chocolates: The non-determinism of ChatGPT in code generation, arXiv:2308.02828, 2023.
15. R. Parasuraman and D. H. Manzey, Complacency and bias in human use of automation: An attentional integration, Human Factors, vol. 52, no. 3, pp. 381-410, 2010.
16. S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, The impact of AI on developer productivity: Evidence from GitHub Copilot, arXiv:2302.06590, 2023.
17. European Parliament and Council, Regulation (EU) 2024/1689 laying down harmonised rules on

© 2026 IJSRET

artificial intelligence (Artificial Intelligence Act), Official Journal of the European Union, 2024.

18. Government of India, The Digital Personal Data Protection Act, 2023.

19. B. Kitchenham and S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE-2007-01, Keele University and Durham University, 2007.

20. M. Popay et al., Guidance on the conduct of narrative synthesis in systematic reviews, ESRC Methods Programme, 2006.

21. X. Hou et al., Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology, 2024.

22. M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, Mutation testing advances: An analysis and survey, Advances in Computers, vol. 112, pp. 275-378, 2019.

© 2026 IJSRET