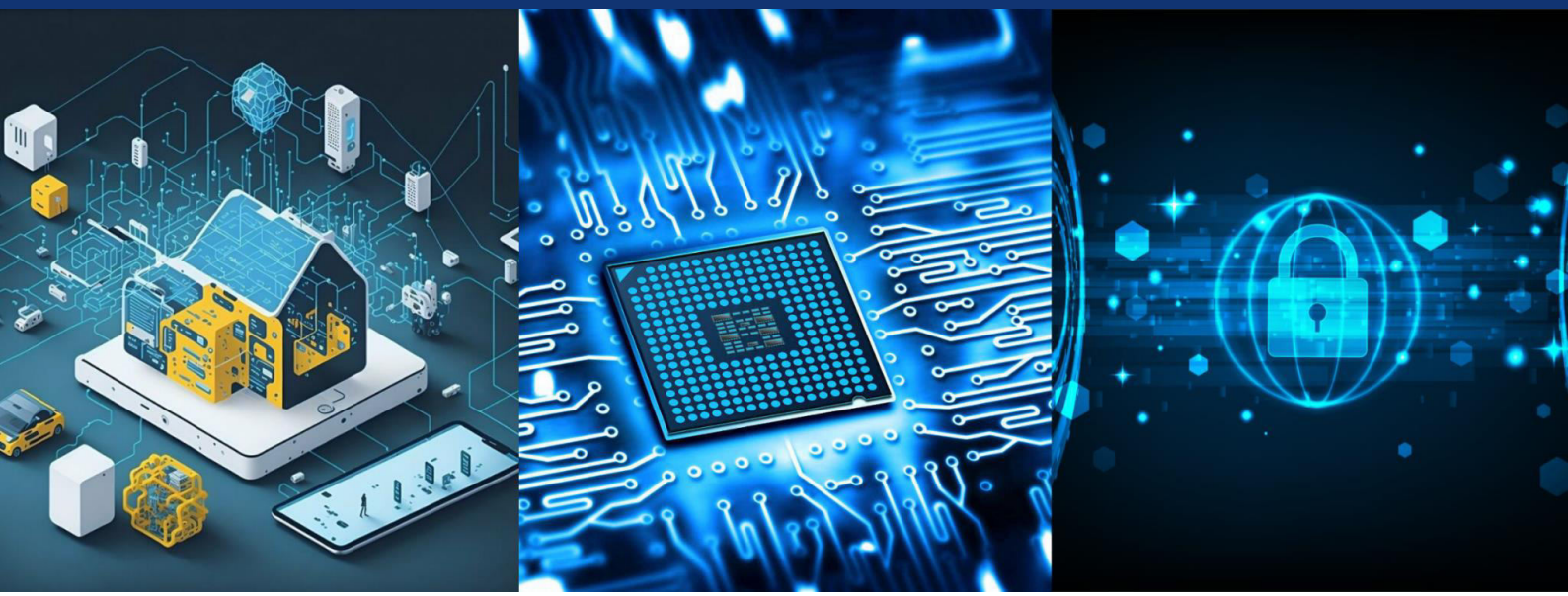
IJIRCCE

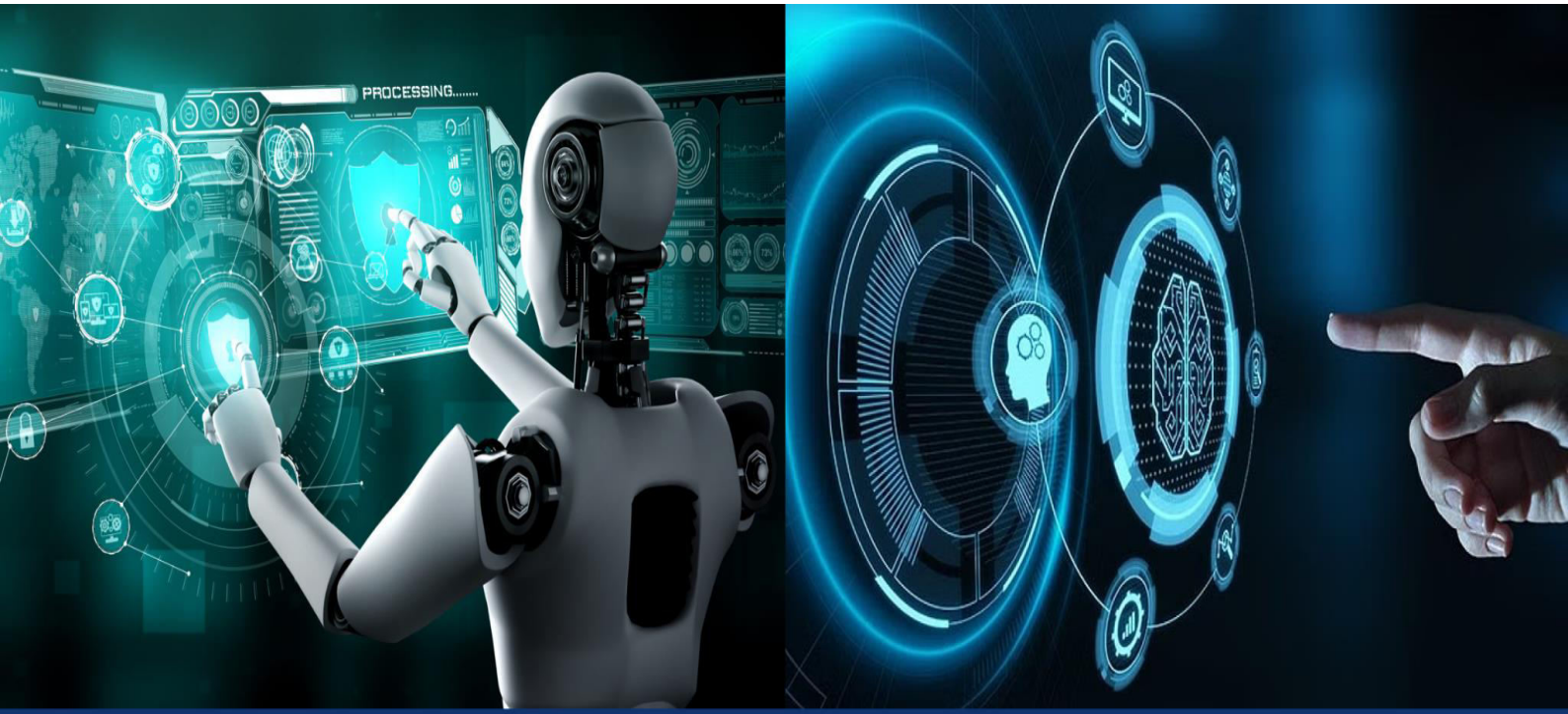
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 10, October 2026

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

AI-Based Automatic Generation of Software Test Cases

Adinath Ramesh Gawande, Mrs. Shubhangi Pratik Bomble

Department of Master of Computer Applications, JCEI's Jaihind Institute of Management and Research, Vadgaon, Sahani, India

Assistant Professor, Department of Master of Computer Applications, JCEI's Jaihind Institute of Management and Research, Vadgaon, Sahani, India

ABSTRACT: Software testing is essential for checking whether a software product behaves according to its requirements and for identifying defects before release. Test cases are commonly prepared from requirement documents, user stories, use cases and acceptance criteria. Manual test design can be time-consuming and may overlook boundary conditions, invalid inputs, role-based permissions or exception scenarios, particularly when requirements change frequently. Artificial intelligence (AI), natural language processing (NLP) and large language models (LLMs) can assist by converting natural-language requirements into structured candidate test cases. This paper reviews AI-assisted test-case generation and proposes a human-supervised framework that includes requirement analysis, generation, validation, deduplication, traceability and reviewer approval. The paper is conceptual: it does not report a deployed prototype, a trained model or experimental performance results. It discusses possible benefits, such as faster drafting and more systematic scenario exploration, together with risks including ambiguous specifications, unsupported assumptions, duplicate tests, privacy concerns and inaccurate expected outcomes. Generated cases should therefore be treated as drafts that require verification against authoritative requirements and, where possible, execution in the application under test. Future evaluation should compare AI-assisted and manual approaches using measures such as requirement coverage, validity, defect detection, duplication and reviewer effort. A descriptive questionnaire with 56 submitted responses is also summarised; the findings reflect respondent perceptions and are not a controlled evaluation of AI-generated test quality.

KEYWORDS: Artificial intelligence, software testing, automatic test-case generation, natural language processing, large language models, test automation, software quality.

I. INTRODUCTION

Software applications support services in education, finance, healthcare, retail, transport and government. As applications grow in complexity, testing becomes increasingly important for reliability, usability and security. A defect can cause incorrect processing, interrupted service, financial loss or reduced user trust. Software testing checks system behaviour against requirements and helps identify failures before release.

A test case typically records a precondition, input data, execution steps and expected result. Testers derive cases from requirements, user stories, use cases, design documents, historical defects and domain knowledge. Manual preparation can be effective, but it takes time when requirements change often or when a system has many user roles and workflows. Important negative cases, boundary values, invalid states and permission checks may be missed.

Automation tools can execute repeatable tests, but test execution automation does not necessarily decide which scenarios should exist. Traditional research has explored model-based testing, search-based generation, symbolic execution and random testing. AI-assisted methods add the possibility of interpreting natural-language requirements and producing candidate scenarios that testers can inspect.

NLP can extract actors, actions, conditions, constraints and expected outcomes from text. Machine-learning methods can identify patterns in historical test repositories, while LLMs can draft structured test cases from prompts and examples. However, a plausible-looking output is not automatically correct: a model may invent a business rule, omit a

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

condition or state an expected result not supported by the source. Human validation and traceability are therefore central to responsible adoption.

This paper reviews the role of AI in automatic test-case generation and proposes a conceptual workflow. It focuses on the design of a controlled process rather than claiming to implement or experimentally validate a new system.

II. PROBLEM STATEMENT

Test-case design can be a labour-intensive activity when requirements contain multiple roles, conditions, exceptions and business rules. Manual approaches may produce inconsistent detail, miss edge cases or lose traceability between a requirement and its tests. If useful test scenarios are not designed, automated execution alone cannot ensure adequate coverage.

AI can generate candidate tests from requirement documents, but it can also create duplicate scenarios, invalid steps, unsupported assumptions or expected results that conflict with the specification. The problem is to design a workflow that helps generate relevant and traceable test cases while controlling generation errors, preserving confidentiality and keeping testers responsible for approval.

III. OBJECTIVES

1. To explain the role of AI, NLP and LLMs in software test-case generation.
2. To review traditional automatic test-generation approaches and emerging AI-assisted methods.
3. To propose a conceptual framework that transforms requirements into candidate test cases.
4. To identify evaluation measures such as requirement coverage, validity, duplication and defect detection.
5. To discuss limitations involving inaccurate outputs, privacy, traceability and human oversight.
6. To outline a future evaluation procedure comparing AI-assisted test design with a manual baseline.

IV. SCOPE AND SIGNIFICANCE

The paper considers candidate functional tests derived from natural-language requirements, user stories and acceptance criteria. Scenarios may include positive paths, invalid inputs, boundary conditions, role-based access and exceptions when these are supported by the specification. The proposed workflow may be adapted for web, mobile or enterprise software when requirements and a test environment are available.

The paper does not claim that AI can replace professional testers or independently certify software quality. No original survey, model training or controlled experiment is reported. Its significance is the description of a practical process that combines AI-assisted drafting with validation, traceability and human approval.

V. LITERATURE REVIEW

5.1 Traditional automatic test generation

Automatic test generation has been studied through several approaches. Model-based testing derives tests from models of system states and transitions. Search-based software testing frames test generation as an optimisation task and searches for inputs that satisfy a coverage objective. Symbolic execution reasons about program paths and constraints, while random and property-based methods explore input spaces. Their usefulness depends on the availability of models, program structure, test oracles and the target coverage objective.

5.2 NLP and requirements-based testing

Natural-language requirements are accessible to stakeholders but may contain ambiguity, incomplete conditions or inconsistent terminology. NLP can help identify actors, actions, objects, conditions and expected outcomes. For example, a password-reset requirement can be decomposed into a user role, a recovery-link condition, an action and an expected result. Yet extraction methods can misinterpret negation, timing, dependencies or domain-specific terms. The original requirement should remain linked to every generated test.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

5.3 Generative AI and large language models

LLMs can produce structured text from prompts and examples. They may draft positive and negative tests, suggest boundary cases and return results in a table or machine-readable format. This can reduce repetitive writing and make test drafting accessible to teams that manage requirements in natural language. However, LLMs can hallucinate interface elements, assume undocumented rules or generate incorrect expected outcomes. Output format and fluency should not be treated as evidence of correctness.

5.4 Human review and test quality

Human review is important because test design depends on business context, risk, user roles and system behaviour. A reviewer can clarify ambiguous requirements, reject unsupported cases and identify missing scenarios. A useful tool should display the source requirement, label uncertainty and allow each candidate to be edited, approved or rejected. Review decisions should be recorded to support auditability and continuous improvement.

5.5 Research gap

Traditional automated testing offers mature techniques for program-based generation, while generative AI creates new possibilities for requirement-based test drafting. A practical challenge is to integrate generation into a traceable workflow and assess not just the number of cases produced but their validity, coverage, usefulness and maintenance cost. This paper addresses that need through a proposed framework and evaluation plan. The framework remains a proposal until implemented and tested.

VI. PROPOSED AI-BASED FRAMEWORK

The proposed framework treats model output as a candidate test artefact, not as an automatically approved test. It accepts requirement text and acceptance criteria, extracts relevant information, generates scenarios, validates their structure and traceability, and presents them to a tester. Approved tests may then be exported to a test-management system or adapted for automated execution.

6.1 Requirement input and preparation

The user provides a requirement, user story, acceptance criteria or selected specification section. The system records the source and version so later changes can be tracked. Confidential documents should be processed only in approved infrastructure; credentials, personal data and proprietary content should not be sent to an external model without authorisation.

6.2 Requirement analysis

The analysis stage identifies actors, preconditions, actions, inputs, constraints and expected outcomes. It can flag vague terms such as “quickly” or “appropriate access” for clarification. If a requirement does not define a measurable outcome, the tool should mark the gap instead of inventing a result. Each extracted element should retain a reference to the source requirement.

6.3 Candidate test generation

A generation stage uses a controlled template to draft positive, negative, boundary, permission and exception cases where justified by the requirements. Instructions should require the model to avoid undocumented behaviour and state uncertainty explicitly. If an assumption is needed, it should be identified for review rather than presented as a confirmed product rule.

6.4 Validation and deduplication

Generated tests should be checked for required fields, logical consistency, traceability and duplication. Structural validation can verify that a case has an identifier, source requirement, preconditions, steps and expected result. Semantic review must determine whether the expected result follows from the specification. Similarity analysis can flag likely duplicates, but a reviewer should make the final decision.

6.5 Review, approval and feedback

A tester reviews each candidate, edits unclear steps and approves only cases that can be justified. The system should record the reviewer, date, source requirement and revision history. Rejection reasons may include duplicate,

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

unsupported assumption, irrelevant scenario or incomplete expected result. Feedback must be managed so confidential information is not exposed.

6.6 Proposed workflow

1. Enter a requirement or user story and record its version.
2. Extract actors, preconditions, actions, constraints and outcomes.
3. Identify ambiguity and request clarification where necessary.
4. Generate candidate positive, negative, boundary and exception tests.
5. Validate structure, source traceability and consistency.
6. Flag likely duplicates and unsupported assumptions.
7. Ask a tester to edit, approve or reject each candidate.
8. Export approved tests and record execution outcomes for evaluation.

VII. ILLUSTRATIVE EXAMPLE

This example illustrates the proposed method and is not a result from an implemented system. Assume the requirement states: “A registered user can reset a forgotten password using a valid, unexpired recovery link. After a successful reset, the user can sign in with the new password.”

TC-04 is valid only if the specification defines a password policy. If no policy is documented, the tool should flag a requirement gap rather than assume a rule. Error messages and token-reuse behaviour should also be included only when documented or agreed with the product owner.

VIII. EVALUATION METHODOLOGY

A future implementation should be evaluated with a documented set of requirements and a reviewed reference set of test cases. The same requirements could be supplied to a manual test-design group and to the AI-assisted workflow. Reviewers should use common acceptance criteria and, where practical, should not know which method generated each case. No experiment was conducted for this paper; this section presents a proposed evaluation plan rather than measured findings.

The evaluation dataset should contain software requirements, functional specifications, input conditions, expected outputs, and relevant boundary conditions. Requirements may be selected from sample applications or publicly available software projects. Each requirement should be reviewed to ensure that it is clear, testable, and sufficiently detailed. A reference set of test cases should be prepared and verified by experienced testers to provide a consistent basis for comparison.

The same set of requirements should be provided to two workflows: manual test-case generation by software testers and AI-assisted test-case generation using an appropriate AI model. Both workflows should follow the same requirement scope and acceptance criteria. This comparison can help identify differences in test-case coverage, preparation time, duplication, and the number of cases requiring correction. Where practical, reviewers should assess test cases without knowing whether they were generated manually or with AI to reduce evaluation bias.

8.1 Suggested measures

Measure	Purpose	Evaluation approach
Requirement coverage	Shows whether each in-scope requirement has tests.	Covered requirements divided by total in-scope requirements.
Test validity	Measures cases judged executable and consistent.	Valid reviewed cases divided by reviewed generated cases.
Scenario coverage	Checks positive, negative and boundary categories.	Compare with a predefined scenario checklist.
Duplicate rate	Identifies repeated or near-identical tests.	Review flagged duplicates among generated cases.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Defect detection	Checks discovery of known or seeded defects.	Compare relevant defects detected under equal conditions.
Reviewer effort	Measures time needed to approve tests.	Record review and editing time.
Traceability	Checks links between tests and requirements.	Cases with verified links divided by total cases.

8.2 Proposed procedure

9. Select representative requirements and define scope in advance.
10. Prepare a reference test set through a documented review process.
11. Generate candidate cases and record model version, prompts and settings.
12. Ask qualified reviewers to assess validity, coverage, duplicates and assumptions.
13. Execute approved cases in the same environment where possible.
14. Compare the AI-assisted workflow with the baseline and report limitations and reviewer effort.

IX. SURVEY STUDY: RESPONDENT PERCEPTIONS

To complement the conceptual framework, a descriptive survey was conducted using the Google Form titled "A Study on AI-Based Automatic Generation of Software Test Cases." The supplied response export contains 56 submissions dated 15–16 October 2026. The analysis below reports the responses present in the spreadsheet; it does not represent a controlled experiment or a measurement of the accuracy of any AI system. One familiarity item has 55 non-empty responses, while the other closed-ended questions have 56. The open-ended suggestion question contains no recorded answers in this export.

9.1 Respondent profile and familiarity

The largest respondent group was Software Tester / QA Engineer (21, 37.5%), followed by Automation Test Engineer (14, 25.0%). Software Developers, IT Professionals and MCA / Computer Science Students were also represented. This profile is relevant because responses largely reflect people with a software or testing context, rather than a general public sample.

For familiarity with software testing, the most common answer was Neutral (28 of 55 valid answers), followed by Slightly familiar (14) and Somewhat familiar (12). As the question has one missing answer and many respondents selected neutral or limited familiarity, the survey should not be interpreted as evidence of expert-level knowledge across the whole sample.

9.2 Manual preparation and current awareness

Manual preparation was reported as Sometimes by 30 respondents (53.6%) and Rarely by 20 (35.7%). Three respondents selected Frequently and three selected Never. This indicates that respondents have differing levels of direct involvement in preparing test cases, so AI tools should support different workflows and experience levels.

Regarding awareness of AI-based automatic generation, 28 respondents said they knew only a little, while 20 had heard of it but did not know details. Six said they understood how it works and two had not heard of it. The pattern suggests that practical demonstrations and clear explanations may be important when introducing such systems.

9.3 Use, perceived usefulness and concerns

For AI tool use, 30 respondents selected Sometimes and 24 selected Tried once or twice; two selected Frequently. Thus, most reported either occasional use or limited experimentation, rather than frequent use. The survey does not identify which tools were used or independently verify their output quality.

Perceived usefulness was most often rated Moderately useful (27; 48.2%), followed by Very useful (16; 28.6%). Five respondents selected Extremely useful and eight Slightly useful. These are opinions about potential usefulness, not evidence that AI generation improves coverage or reduces effort in practice.

The most frequently selected concern was Missing important scenarios (19), followed by Incorrect expected results (18) and Duplicate test cases (12). These concerns align with the proposed framework's emphasis on coverage checks, expected-result validation, deduplication and explicit human approval.

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

9.4 Human review and desired system features

For whether human testers should review AI-generated cases before approval, 31 selected Usually, 15 Sometimes, 6 Rarely and four Always. Although the response distribution is not unanimous, the largest group favoured review usually being required. This supports a human-in-the-loop design in which generated cases remain drafts until assessed against requirements.

The most selected feature was Positive, negative and boundary scenario coverage (23), followed by Duplicate detection (14) and Easy editing and human approval (8). Requirement traceability and privacy/data security were also selected. The feature priorities are consistent with the proposed design: useful generation must be paired with quality checks, editable outputs and a traceable approval process.

9.5 Selected response summary

Survey item	Most common response	Count	% of valid answers
Respondent role	Software Tester / QA Engineer	21	37.5%
Respondent role	Automation Test Engineer	14	25.0%
Respondent role	Software Developer	9	16.1%
Awareness of AI-based generation	Yes, but I know only a little	28	50.0%
Awareness of AI-based generation	I have heard of it but do not know details	20	35.7%
Awareness of AI-based generation	Yes, and I understand how it works	6	10.7%
Use of AI tools for test cases	Sometimes	30	53.6%
Use of AI tools for test cases	Tried once or twice	24	42.9%
Use of AI tools for test cases	Frequently	2	3.6%
Perceived usefulness	Moderately useful	27	48.2%
Perceived usefulness	Very useful	16	28.6%
Perceived usefulness	Slightly useful	8	14.3%
Concerns about AI-generated tests	Missing important scenarios	19	33.9%
Concerns about AI-generated tests	Incorrect expected results	18	32.1%
Concerns about AI-generated tests	Duplicate test cases	12	21.4%
Human review before approval	Usually	31	55.4%
Human review before approval	Sometimes	15	26.8%
Human review before approval	Rarely	6	10.7%
Important system features	Positive, negative and boundary scenario coverage	23	41.1%
Important system features	Duplicate detection	14	25.0%

9.6 Charts of selected survey items

The following charts summarise selected closed-ended questions from the supplied export. Percentages are calculated using non-empty answers for each item; counts are descriptive and should be interpreted in light of the sample size and recruitment method.

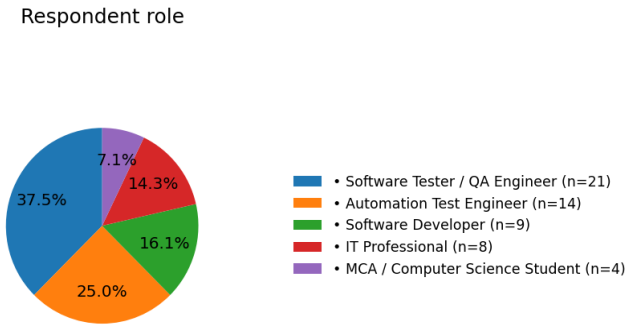
© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

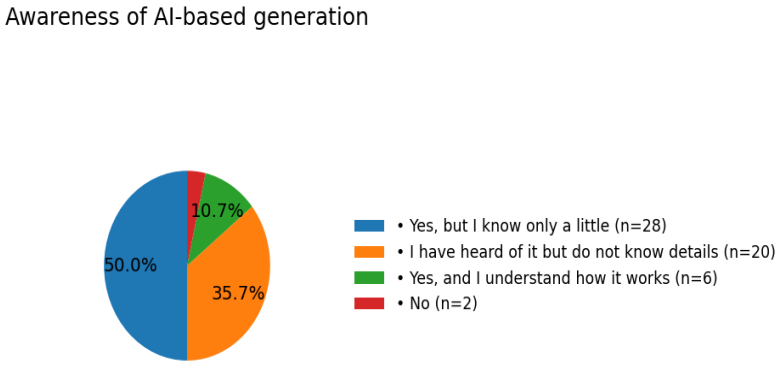
International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

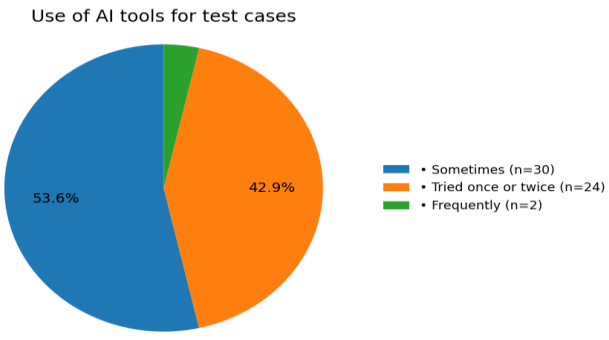
Respondent role



Awareness of AI-based generation



Use of AI tools for test cases



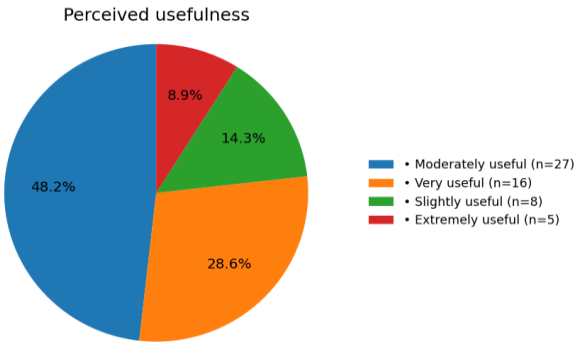
© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

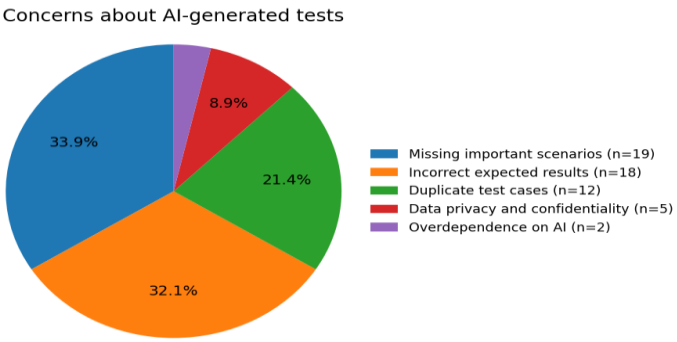
International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

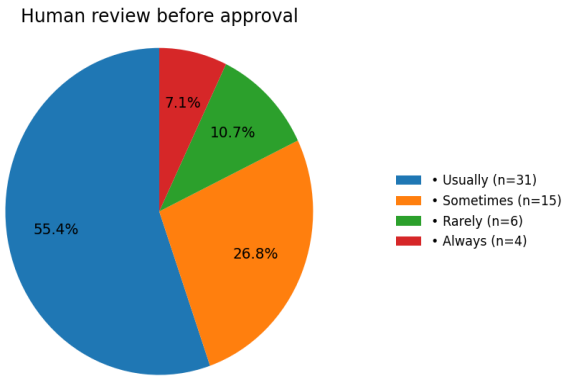
Perceived usefulness



Concerns about AI-generated tests



Human review before approval



9.7 Interpretation of survey findings

Taken together, the responses indicate interest in AI-assisted test-case drafting alongside caution about omissions, incorrect expected results and duplication. Respondents identified boundary-case identification and test-data generation as useful capabilities, while coverage, duplicate detection and human approval emerged as important system features. These findings are consistent with—but do not validate—the conceptual framework presented in this paper. A future study should recruit a clearly described sample, test an implemented prototype against a manually reviewed baseline, and measure validity, requirement coverage, duplicate rate, reviewer effort and defect detection.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

9.8 Survey limitations and ethical reporting

The survey is descriptive and based on a convenience response set; the spreadsheet does not document the sampling frame, recruitment method, respondent demographics beyond role, or whether each respondent represents a distinct individual. Results may therefore not generalise to all software professionals. Several items were single-choice in the export, and the open-ended suggestion field has no recorded responses. No causal claim can be made from these results. Before submission, the author should confirm consent and institutional requirements for using survey responses, remove any personal information not required for analysis, and retain the original response file securely. No fabricated responses have been added.

X. DISCUSSION

AI-assisted generation may reduce time spent drafting routine scenarios and help testers explore a wider range of inputs. It may be useful when requirements are maintained as structured user stories or acceptance criteria. Structured outputs can also simplify transfer into test-management tools. These benefits depend on input quality and careful review.

A major risk is that a generated case sounds plausible but contradicts the specification. A model may fill gaps with common assumptions or overlook a condition in a long requirement. Traceability, explicit uncertainty and human approval help reduce this risk. When information is missing, the tool should request clarification rather than invent an expected result.

The number of generated cases is not a reliable quality measure by itself. Large outputs may contain duplicates, trivial variations or cases that cannot be executed. Validity, requirement coverage, defect detection and reviewer effort provide a more meaningful assessment. Test maintenance also matters: changes to requirements should trigger review of linked cases.

Security and privacy must be considered in enterprise environments. Requirements can contain business-sensitive information, and test cases can expose internal workflows. Organisations should define approved model providers, access controls, retention limits and audit practices. Generated tests should not include real passwords, personal information or production secrets.

AI is best treated as a support tool for testers. It can assist with drafting and scenario variation, while testers remain responsible for relevance, expected behaviour and approval.

XI. LIMITATIONS

This paper is a literature-informed conceptual study. It does not report a newly trained model, deployed application, experimental dataset or measured performance. The illustrative test cases explain the proposed workflow and do not prove that an AI tool generated or executed them. Reviewed approaches also differ in inputs, systems and evaluation methods, so results from different studies may not be directly comparable.

The framework needs implementation and evaluation before productivity, coverage or defect-detection claims can be made. Future work should document model and prompt versions, use representative requirements, define reviewer criteria in advance and report both positive and negative findings.

XII. CONCLUSION AND FUTURE WORK

AI-based automatic generation of software test cases can assist test engineers in transforming natural-language requirements into structured candidate tests. NLP can identify actors, actions and conditions, while generative models can draft positive, negative, boundary and exception scenarios. However, fluent output is not necessarily valid, and undocumented assumptions can lead to incorrect expected results.

This paper proposed a human-supervised framework consisting of requirement preparation, extraction, generation, validation, deduplication, reviewer approval and feedback. It also outlined an evaluation plan based on coverage,

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

validity, scenario coverage, duplicate rate, defect detection, traceability and reviewer effort. Since no implementation or experiment was conducted, the framework should be understood as a proposal rather than a validated system.

Future work can implement a prototype, compare models and prompts, and evaluate generated tests against a manually reviewed baseline. Further study should address ambiguity, maintenance after requirement changes, privacy-preserving deployment and integration with test-management and continuous-integration tools. Responsible use should keep testers in control and make each test traceable to an authoritative requirement.

XIII. ACKNOWLEDGMENT

The author acknowledges the guidance of Mrs. Shubhangi Pratik Bomble and the academic support of the Department of Master of Computer Application, JCEI's Jaihind Institute of Management and Research, if this accurately reflects the support received.

REFERENCES

[1] P. McMinn, “Search-based software test data generation: A survey,” Software Testing, Verification and Reliability, vol. 14, no. 2, pp. 105–156, 2004. doi: 10.1002/stvr.294.
[2] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in Future of Software Engineering (FOSE '07), 2007, pp. 85–103. doi: 10.1109/FOSE.2007.25.
[3] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” Software Testing, Verification and Reliability, vol. 22, no. 5, pp. 297–312, 2012. doi: 10.1002/stvr.456.
[4] M. Harman, “The current state and future of search based software engineering,” in Future of Software Engineering, 2007, pp. 342–357. doi: 10.1109/FOSE.2007.29.
[5] A. Fraser and A. Arcuri, “EvoSuite: Automatic test suite generation for object-oriented software,” in Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2011, pp. 416–419. doi: 10.1145/2025113.2025179.
[6] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach. San Francisco: Morgan Kaufmann, 2007.
[7] ISO/IEC, ISO/IEC 25023:2016, Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Measurement of System and Software Product Quality, 2016.
[8] IEEE, IEEE Standard for Software and System Test Documentation, IEEE Std 829-2008, 2008.
[9] A. Mesbah, A. van Deursen, and S. Lenselink, “Crawling Ajax-based web applications through dynamic analysis of user interface state changes,” ACM Transactions on the Web, vol. 6, no. 1, article 3, 2012. doi: 10.1145/2109205.2109208.
[10] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023. doi: 10.6028/NIST.AI.100-1.
[11] 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, doi: 10.1109/TSE.2024.3368208.
[12] M. Fontes, M. d’Amorim, and co-authors, “The integration of machine learning into automated test generation: A systematic mapping study,” Software Testing, Verification and Reliability, vol. 33, no. 4, e1845, 2023, doi: 10.1002/stvr.1845.
[13] “Effective test generation using pre-trained Large Language Models and mutation testing,” Information and Software Technology, vol. 171, article 107468, 2024, doi: 10.1016/j.infsof.2024.107468.
[14] M. Baqar and R. Khanda, “The Future of Software Testing: AI-Powered Test Case Generation and Validation,” arXiv:2409.05808, 2024.
[15] Y. Chen et al., “A survey of unit test case generation methods,” Information and Software Technology, 2026



INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details