COLOMBO SCIENTIFIC PUBLISHING

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE & INFORMATION SYSTEM
E-ISSN: 2536-7919
P-ISSN: 2536-7900
PAGE NO: 11-19

A Review of NLP-Driven Test Case Generation Using Large Language Models for Software Quality Assurance

Dr. Nilesh Jain
Associate Professor Department of Computer Sciences and Applications, Mandsaur University, Mandsaur

ARTICLE INFO

Article history:
Submission: October 28, 2026
Accepted: October 07, 2026
Published: November 18, 2026
VOLUME: Vol.11 Issue 11 2026
DOI: -
10.55640/ijcsis/Volume11Issue11-02

Keywords:
Large Language Models (LLMs), Automated Test Case Generation, Natural Language Processing (NLP), Software Quality Assurance (SQA), AI-driven Testing.

ABSTRACT

The rapid transformation of software development practices, from Waterfall to Agile and DevOps, has augmented the demand for effective, scalable and precise testing methodology. Conventional manual test case generation is lengthy, inaccurate and in most cases unable to meet current continuous delivery cycles. Recently, advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) have presented exciting opportunities for generating automated test cases. This survey discusses how NLP-based and LLM-based methods can transform the nature of software quality assurance by examining the requirement documents, categorizing intents, creating formalized test cases, and validating the results using automated simulation frameworks. An overview of the architectures of software quality assurance its process and its importance in development is also outlined extensively. Recent literature has shown great improvements including higher coverage of tests, high defect detection rates, and less manual work. Nevertheless, the issues of hallucination, data quality restrictions, computational costs, and the inability to test dynamic interfaces remain. The present review summarizes the existing knowledge, identifies a gap in literature, and provides knowledge of the future of automated generation of test cases with intelligence, scalability, and reliability.

I. INTRODUCTION

The software industry has transitioned from the inflexible Waterfall paradigm to the contemporary Agile and DevOps methodologies over several decades [1]. This has required swift, continuous, and reliable testing methodologies. Software Quality Assurance (SQA) encompasses a series of activities that guarantee a software development product aligns with the specified quality standards [2]. It is an important process that determines the reliability, convenience, and productivity of the software once it has been released. Traditional ad hoc testing procedures may be labor-intensive, expensive, and time-consuming. Automated Testing has emerged to be the new efficient approach of addressing all these requirements. Test automation is the habit of performing tests automatically, test data and using the results in order to enhance software quality [3][4]. Efforts are being made by the whole software production team, with a primary focus on quality assurance. Incorporating all relevant parties, including business analysts, developers, and DevOps engineers, is essential for optimizing test automation [5]. The evolution of software and other forms of information technology has grown ubiquitous. Since IT systems and software could affect millions of people and companies, ensuring their security and dependability is of the utmost importance [6] There always be mistakes and flaws in software developed with human participation.

Natural language processing (NLP) is an area of AI and linguistics concerned with how to train computers to understand human speech. The need to simplify user tasks and fulfill the need to converse with computers in a natural way gave rise to natural language processing [7]. The advent of LLMs can be attributed to advancements in deep learning methods, the availability of massive computing resources, and ample training data. This class of language models takes advantage of self-supervised learning to train neural networks with billions of parameters on large datasets of unlabeled text. It is considered a huge step forward by many in the AI and NLP communities [8]. The practice of software testing is vital, well-liked, and costly since it ensures that a software system is of excellent quality. Despite significant advancements in test automation, manual testing is still heavily used in the software industry. The development team and QA engineers spend more time and effort on testing and test case design when manual testing is performed, which raises the cost of testing for the business [9][10]. The use of NLP techniques provides optimism for the resolution of these problems. Software developers increasingly turn to NLP approaches to automate the generation of requirements, design artifacts, and test cases from raw data.

A. Structure of the Paper

The paper is structured as follows: Section II covers software quality assurance in general, Section III explains natural language processing (NLP) based test case generation, Section IV discusses large language model (LLM) based test case

generation, Section V discusses the difficulties of automated test case generation, Section VI gives a detailed review of recent research on quality assurance protocols in NLP-driven test case generation using large language models, and Section VII gives a general conclusion about test case generation.

II. SOFTWARE QUALITY ASSURANCE

Software Quality Assurance (SQA) is a process-based and structured method that assures software quality that matches the specified quality standards during the development lifecycle. SQA minimizes costs, enhances reliability and prevents production failures by detecting defects at early stages. It increases customer satisfaction, aids in adherence to industry rules, and enables continuous improvement. SQA has enabled the realization of a stable environment in the Agile and DevOps setting by providing automation and continuous testing and feedback on quality, resulting in reliable software.

A. Importance of Software Quality Assurance

SQA is used to produce reliable, bug-free software by eliminating bugs at early stages, minimizing costs, improving performance, adhering to standards, and increasing customer satisfaction during the SDLC.

1) Ensuring Software Quality and Reliability

The most significant component of soliciting, assessing, and enforcing the quality of the software is SQA. Systematic testing and quality control procedures are utilized by SQA to test the software's behavior under numerous circumstances. This data is helpful for finding problems at their earliest stages. When it comes to software, reliability is of the utmost importance because mistakes can result in harm, lost revenue, and a damaged reputation for the company.

2) Reducing Costs and Time

The SQA streamlines the process of identifying costly mistakes and issues. The time and money needed to fix a problem decreases when caught early on. Consequently, the S-SDLC's emphasis on quality helps the process run more smoothly and saves money in the end [11][12].

3) Enhancing Customer Satisfaction

The main objective of SQA is to ensure that the product is of the highest quality in order to meet the expectations of the consumer. After determining that all of the software's functions are functioning correctly, functionality testing ensures that the program can run at its best without any security flaws, and finally, it finds and fixes all of the software's problems.

4) Meeting Regulatory and Compliance Standards

The software industry also has its own set of regulations and compliance requirements that developers must adhere to. Data or information security, privacy, and performance are only a few areas that could be governed by such rules [13]. This is because software quality assurance continuously checks the program to make sure it follows all the rules.

5) Facilitating Continuous Improvement

Quality Assurance in software development is an ongoing procedure, not an isolated incident. Testing and analyzing performance data to enhance software is, hence, what software quality assurance is all about. The product should evolve with the user and be able to incorporate new technology; QA teams are aware of where to find these opportunities and how to assess patterns. Design, coding, and testing are all enhanced over time as a result of quality assurance testing, which also offers suggestions that can influence the development plan [14].

6) Supporting Agile and DevOps Methodologies

Methodologies like Agile and DevOps place a premium on fast iteration, tight cooperation, and frequent status updates in software development [15]. And so, these approaches—backed by SQA—strive to maintain software stability and quality amidst fast development [3][16]. These approaches prioritize early feedback delivery for defects and regressions through testing automation [17], continuous integration, and performance monitoring.

B. Process of Software Quality Assurance

SQA method is built on a certain lifecycle that evaluates all the phases of the software development process [18]. It consists of appropriate planning and data collection which may be in the form of various documents. These documents may include SQA strategy, project management plan or pertinent project metrics and these are usually stored in project documentation store. SQA is an approach, not a process, that guarantees dependability and credibility in all project management and SDLC operations. Figure 1, shows the relationship between the SQA method and the development and project management stages.

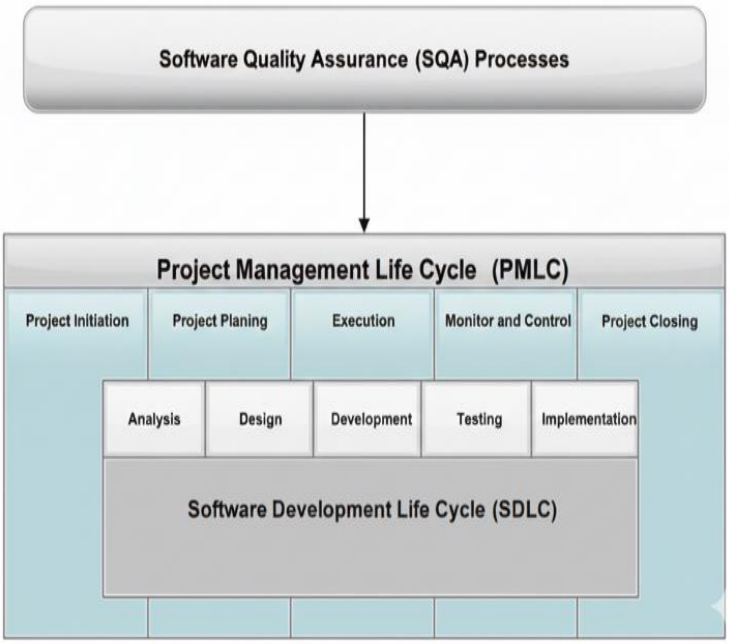


Fig. 1. Process of Software Quality Assurance (SQA)[19]

Figure 1 shows that Software Quality Assurance (SQA) processes are all-inclusive and lead to the entire Project Management Life Cycle (PMLC). Project Management Body of Knowledge (PMBOK) phases include starting a project, making plans, carrying them out, checking in on progress, and finally, wrapping up. The execution phase showcases the SDLC, which encompasses analysis, design, development, testing, and implementation. On the whole, the diagram highlights that SQA activities cut across project management as well as software development in order to deliver quality results.

III. NLP-BASED TEST CASE GENERATION FOR SOFTWARE QUALITY ASSURANCE

The framework, with the help of semantic analysis, is able to consider the context of the sentences in the requirement documents and thus can correctly identify both non-trivial and trivial scenarios [20].

A. Contextual Embedding using Transformers:

The meaning of the required text can be better represented with contextual embedding, especially when using transformers like BERT or T5. Transformers give contextual representations that change with surrounding words, in contrast to normal embeddings' single representation for each word [21].

- Process: Transformer models, like BERT, are fine-tuned using domain-specific datasets so that they can identify software-specific language and generate concepts that mimic more complicated sentences with structures.
- Example: The requirement would be rendered nonsensical due to the BERT-based embedding's interpretation of the terms "lock" and "failed attempts to log in" as referring to security and triggering circumstances, respectively.

Since transformer models have a holistic view of sentences, including all tokens, they are able to learn not only sequential dependencies but also dependencies/conditions that a rule-based approach could overlook. This makes them understand the situation in a broader context.

B. Classification-Based Intent Recognition:

The purpose categorization helps establish the intended objective or goal for each requirement, which in turn makes test case generation more relevant and accurate [22].

- Process: A custom-trained model with labeled data is used to construct classification models for requirements. For example, it can sort statements according to their intended use (login, error handling, transaction verification, etc.).

As an example, when the statement is "Generate a confirmation email, if the payment is successful," the model understands the intent as "post-payment confirmation" and supplies inspired test conditions related to that aim.

C. Templates Matching with Rule-Based Patterns:

Template matching takes the organized output from NER and Dependency Parsing and maps it to test case templates that have already been set up. For simpler automation and integration into the testing system, this ensures that all test cases generated from sentences have the same format for instance. The user's login is successfully processed: IF THEN; corresponds to Condition and Action. The testing teams have an easier time reading and maintaining the test cases because they all adhere to the same format.

D. Rule-Based Post-Processing and Validation

This ensures that the test cases not only meet the requirements standards but also project expectations by applying domain-specific rules to the result set in the following stage, rule-based processing [23].

- Domain-Specific Rule Refinement: Regulatory compliance in fields like healthcare and finance necessitates the development of industry-specific norms.
- For example, there should be suitable access requirements and/or error messages provided following failed activities involving sensitive data in a banking application, for example while checking an account balance [24].

The project's unique set of rules is iteratively developed in response to input from the testing teams; these rules are then put into place to weed out any irrelevant or superfluous cases.

E. Simulator-Driven Automated Validation

A number of automated test runs are carried out to guarantee that the created test cases accurately depict the anticipated behavior. For instance, using Selenium or Robot Framework to model test cases in an artificial environment.

- Benefits: Real application behavior and test cases that were made are compared, and any changes are pointed out. This gives immediate feedback on how to improve the NLP pipeline's use.
- E.g. requirement: The expected errors should be present, and the tested cases make sure that they do in application contexts by triggering an error message. Additionally, the tested cases check if the documentation of requirements matches up with the execution of tests.

IV. ROLE OF LLMS IN ENHANCING SOFTWARE QUALITY ASSURANCE

SQA that makes use of large language models helps to simplify test case preparation by automating requirements interpretation, improving problem identification, and expanding coverage. In reality, this is quite an effective method of meeting the set SQA objectives of reliability, maintainability, and accuracy of validation.

The quality of software needs to be maintained by efficient and reliable testing [25]. The common test case generation techniques in the past are generally manual and are labor-intensive and subject to human error. This is all changing thanks to LLMs, which automate test cases, boost test accuracy, and shorten the process [26][27].

There are several ways in which software testing can be made more efficient and effective by using LLMs to generate test cases. LLMs lessen the coverage, lower the frequency of mistakes, and ease the manual strain of testing. When it comes to Test Case Generation, the main benefits of using LLMs are:

- Automation: Create requirement-, user-based and code-generated test cases.
- Improved Coverage: Detect edge cases and complicated situations that are not detected manually.
- Faster Execution: Automate the test creation and execution by AI.
- Error Reduction: Reduce human tasks in designing and scripting tests.
- Cost Efficiency: Reduce resource-intensive testing phases.
- Continuous Learning: Improve test performance based on past test performance:

LLMs are capable of generating numerous and different test cases [28] with high efficiency and coverage of tests. Conventional testing techniques tend to overlook edge cases, whereas LLMs analyze application data, code and previous test outcomes to create overall test scenarios [29]. This not only ensures wider coverage, it also speeds up the testing process saving manual work and minimizing mistakes. The important ways that LLMs help to improve test coverage and efficiency include:

- Comprehensive Test Generation: Functional, edge and boundary test cases are generated automatically.
- Scenario Identification: Design test scenarios He/she should design realistic test scenarios with both common and rare usage.
- Automated Code Analysis: Create suitable test cases by analyzing source code for vulnerabilities.
- Faster Test Creation: Reduce the need for human interaction and speed up the test-generating process.
- Gap Detection: Identify areas where testing is lacking and provide new scenarios to cover.
- Smart Prioritization: Use application risk and past data to prioritize important test cases.
- Continuous Improvement: Increase precision by refining test cases using historical test data.

- Reduced Redundancy: Reduce the number of unnecessary or duplicate tests.
- Enhanced Test Data Generation: Create realistic test data to enhance the effectiveness of testing.
- Seamless Integration: Create automated testing pipelines that leverage LLM-powered technologies.

Table I below describes several kinds of test cases that LLMs can produce, as well as their respective uses:

TABLE I. TYPES OF TEST CASES LLMS CAN GENERATE

Type of Test Case	Description	Purpose
Functional Test Cases	Validate application functionality against requirements.	Ensure the software works as intended.
Regression Test Cases	Re-test existing functionality after updates.	Detect unintended changes or bugs.
Edge Test Cases	Test boundary values and edge scenarios.	Identify issues in extreme conditions.
Negative Test Cases	Provide invalid inputs to check system responses.	Ensure the application handles errors gracefully.
Performance Test Cases	Evaluate system speed, stability, and scalability.	Assess the application under load and stress.
Security Test Cases	Simulate attacks to identify vulnerabilities.	Serenity.
Integration Test Cases	Test interactions between modules or external systems.	Ensure seamless data flow.
Exploratory Test Cases	Unstructured testing based on intuition and experience.	Discover unexpected bugs.

V. CHALLENGES IN AUTOMATED TEST CASE GENERATION

The implementation of AI-driven automated test case generation, which is very desirable and advantageous, is contingent upon overcoming a number of challenges [30]. Several issues necessitate resolution, including data quality, model training, resource requirements, and the intricacy of implementing AI-based testing across various software platforms.

A. Data Quality and Model Training

AI models rely on execution logs, reports for defects, historical test cases, and test cases to create meaningful test cases. However, erroneous test cases generated and the framework's effectiveness diminished due to a lack of accessible or high-quality data.

Additionally, test generation could be skewed if there are uneven datasets with an abundance of certain defect kinds. Additionally, in order to enhance their software applications, training AI models requires ongoing upgrades [31]. In the absence of periodic retraining, models would generate useless or out-of-date test cases. The establishment of an automatic feedback loop to refine the models is fundamental to any learnable system; nevertheless, this adds complexity to the framework.

B. Computational Resources

AI-driven test generation, particularly for DL or RL-based models, requires a lot of processing power due to its 'big data' requirements, which makes it difficult to generate test cases efficiently [32][33]. Computationally intensive tasks like training these models necessitate state-of-the-art hardware like GPUs or cloud-based infrastructure.

However, firms that lack sufficient resources may find it tough to deploy AI-based testing. The goal of automated testing is to validate software more quickly, but there is a risk that tests run behind schedule if use complex AI models. However, there is still no permanent solution to the problem of how to balance speed and precision in AI-driven software testing.

C. Complexity of Software Systems

Automated test case generation using artificial intelligence is challenging when dealing with complex or even older systems. Even while AI-based frameworks are all the rage now, they might not be compatible with older software applications that were developed without enough documentation. And because real-time systems are inherently unpredictable, AI models may struggle to adequately capture their dynamic character. Clarifying AI-generated test scenarios is a challenge. Tests created by hand may not be interpretable, and developers may have trouble validating their meaning, therefore there's no use in doing so. As a result, AI-driven testing can be more openly implemented.

VI. LITERATURE REVIEW

This section contains a full explanation of the study, Role of LLMs in enhancing software quality assurance, as well as a condensed version of the information in Table II.

X. Zhao et al. (2026) explore the integration of Artificial Intelligence (AI) and automation within the Behavior-Driven Development (BDD) paradigm, using the PyTest-BDD framework, to enhance Software Quality Assurance (SQA) processes.

This approach fosters a collaborative environment between human testers and AI agents, leading to more robust testing with reduced human overhead [34].

Li et al. (2025) suggest a technology that uses artificial intelligence to generate test cases, using the DeepSeek-Chat big model. This technology would improve software testing efficiency and level of automation by streamlining the process from requirements to test case generation. To improve test coverage and defect detection rates, DeepSeek-Chat uses ML and NLP to automatically generate high-quality test cases. Research shows that this technology increases the rate of problem detection by 25% and the coverage of tests by 30% when compared to conventional methods [35].

Abideen and Junxia (2025) propose a novel approach named E2E-TestGen, which based on intent-driven testing, leverages LLMs. E2E-TestGen interacts dynamically with the DOM by triggering JavaScript-driven updates and tracking real-time state transitions, ensuring comprehensive validation across all dynamic elements and interactions to improve coverage. For canvas-rendered graphical components, they integrate an image-based object detection and text extraction module to extract meaningful data from visual content, enabling the generation of realistic test case scenarios guided by user intent. Experiments on multiple open-source web applications are conducted, which show E2E-TestGen achieving 88.5% and 80.1% code and branch coverage, respectively, along with a 27.91% improvement in fault detection over baseline approaches [36].

Naimi et al. (2024) suggest a new framework that uses generative AI to turn use case diagrams into test cases that can be executed. By getting information from the XML representation of use case diagrams, they can create detailed instructions that guide a generative AI model to make test cases for each use case scenario. This approach not only simplifies the process of creating test cases but also makes them comprehensive and accurate, potentially leading to software products reaching the market more quickly. This approach demonstrates the potential of traditional software engineering methods and novel AI technology to co-exist, providing us an idea of what automated software testing could look like in the future [37].

Edirisinghe and Wickramaarachchi (2024) explore QA for LLM-generated test cases in black-box testing through a systematic literature review. Though LLMs are increasingly used for test case generation, challenges in ensuring their quality remain. Following PRISMA guidelines, relevant studies were selected from databases focusing on critical quality attributes, QA frameworks, metrics, and challenges. LLMs demonstrate high efficiency but face numerous issues. A recommendation for future research is given on addressing standardized metrics and improving human-AI collaboration for enhanced testing outcomes [38].

Chen, Zhao and Li (2023) created the program starting with the SysML model as an input, built the software activity digraph and then created the operational black box test profile. An optimized test sequence is generated from the profile using the traversal approach, and testing efficiency is further improved. In the end, the elevator control software is selected for verification. A SysML model and test profile are established. Then, test sequence reduction is used to produce effective test cases [39].

TABLE II. SUMMARY OF RECENT STUDIES ON NLP-DRIVEN TEST CASE GENERATION USING LARGE LANGUAGE MODELS

Source	Research Focus	Key Insights	Challenges	Limitations
Zhao et al. (2026)	Integration of Artificial Intelligence and automation into Behavior-Driven Development (BDD) using the PyTest-BDD framework.	Introduces human–AI collaboration for automated software testing, improving test robustness while reducing manual testing effort.	Effective coordination between human testers and AI agents, maintaining the accuracy of generated tests, and integrating AI into existing BDD workflows.	Limited information on large-scale validation, performance comparisons, and the generalizability of the approach across different testing frameworks.
Li et al. (2025)	AI-driven test case generation using the DeepSeek-Chat large language model, transforming software requirements into test cases.	Reports a 25% improvement in problem detection and a 30% increase in test coverage compared with conventional approaches through ML- and NLP-based test generation.	Ensuring the semantic accuracy of generated test cases, interpreting ambiguous requirements, and maintaining consistency across complex testing scenarios.	The reported improvements may depend on the datasets and experimental settings. Further validation across different software domains and LLM architectures is required.
Abideen and Junxia (2025)	Development of E2E-TestGen, an intent-driven end-to-end testing approach using LLMs, dynamic DOM interaction, and image-based object detection.	Achieves 88.5% code coverage, 80.1% branch coverage, and a 27.91% improvement in fault detection over baseline approaches.	Handling dynamic DOM updates, JavaScript-driven state transitions, and extracting meaningful information from canvas-rendered graphical components.	Performance may vary across complex web applications and visually intensive interfaces. Additional evaluation is needed for scalability and cross-platform applicability.
Naimi et al. (2024)	Generative AI-based transformation of use case diagrams into executable software test cases using XML representations.	Demonstrates how structured UML use case information can guide generative AI in producing detailed test cases for different use case scenarios, reducing manual test design effort.	Extracting accurate information from XML-based use case diagrams, interpreting complex scenarios, and ensuring the completeness and executability of generated test cases.	The approach relies on structured use case diagrams and may not fully address unstructured requirements, evolving specifications, or complex real-world testing environments.

Edirisinghe and Wickramaarachchi (2024)	Systematic literature review of quality assurance for LLM-generated test cases in black-box software testing.	Identifies the efficiency of LLM-based test generation while highlighting the importance of standardized quality attributes, evaluation metrics, and human–AI collaboration.	It establishes reliable quality assessment frameworks, measuring test case correctness, and addressing inconsistencies in existing evaluation methods.	As a systematic literature review, it does not introduce or experimentally validate a new test-generation framework. Its findings also depend on the availability and quality of existing studies.
Chen, Zhao, and Li (2023)	Model-based black-box test generation using SysML activity models, software activity digraphs, and optimized test sequences.	Develops an approach for generating and reducing test sequences, demonstrated through elevator control software verification, to improve testing efficiency.	Constructing accurate activity digraphs, optimizing test sequence traversal, and maintaining adequate test coverage while reducing redundant test cases.	The approach primarily relies on model-based engineering rather than NLP or LLM-driven test generation. Its applicability to unstructured requirements and AI-generated test cases remains unexplored.

VII. CONCLUSION AND FUTURE WORK

Software quality assurance has taken a giant leap forward with the use of natural language processing and linear logic for the automated creation of test cases. Through contextual knowledge, intent detection, systemic mapping of templates and automatic validation, it has been demonstrated that LLMs provide the ability to reduce the number of manual entries, enhance accuracy and provide a better test coverage. One of the quantifiable improvements that can be proved with references to the literature review is the increased rate of defect identification and the more efficient testing processes in different spheres of software. Despite such developments, the existing systems have not addressed issues such as hallucinations, poor domain generalization, high computing requirements and complicated or dynamic interfaces. Moreover, the method of evaluation is challenging to do reliably since there are no standardized measures of evaluation to evaluate the test cases generated by LLMs. Further studies are needed to mitigate hallucinations in the test cases generated by LLMs by enhancing grounding, domain-specific fine-tuning, and multimodal validation. They need more standardized and universally acceptable measures of evaluation in order to measure reliability, consistency and functional correctness. Further AI-generated test case improvement can be done by having human-in-the-loop supervision and adaptive feedback loops. Additionally, it will render the NLP-based automated testing useful in the industry by increasing the number of approaches to support real-time systems, large enterprise applications and dynamic UIs.

REFERENCES

1. V. Banka, “Multi-Dimensional Governance in Software Procurement: A Strategic Framework for Integrated Asset Management, Cybersecurity, and Legal Compliance Within the ServiceNow Ecosystem,” in 2026 7th International Conference On Computational Vision and Bio Inspired Computing (ICCVBIC), IEEE, Aug. 2026, pp. 1271–1277. doi: 10.1109/ICCVBIC71195.2026.11689058.
2. S. Bernardo et al., “Software Quality Assurance as a Service: Encompassing the quality assessment of software and services,” Futur. Gener. Comput. Syst., vol. 156, no. February, pp. 254–268, Jul. 2024, doi: 10.1016/j.future.2024.03.024.
3. S. Barman, P. Gupta, and S. Kashiramka, “Project Management Survey: A Review of Software Project Management Methodologies,” 2024 IEEE 11th Uttar Pradesh Sect. Int. Conf. Electr. Electron. Comput. Eng. UPCON 2024, 2024, doi: 10.1109/UPCON62832.2024.10983518.
4. K. Patel and S. Gupta, “Prediction of Software Defects for Quality Assurance and Improvement Based on Machine Learning Methods,” in 2024 2nd International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT), IEEE, Nov. 2024, pp. 400–406. doi: 10.1109/ICAICCIT64383.2024.10912382.
5. Y. Macha and S. K. Pulichikkunnu, “A Survey of DevOps Practices for Machine Learning and Artificial Intelligence Workflows in Modern Software Development,” ESP J. Eng. Technol. Adv., vol. 4, no. 3, pp. 200–208, 2024, doi: 10.56472/25832646/JETA-V4I3P121.
6. H. Afzal and M. Huzaifa, “Securing AI Systems Against Adversarial Attacks: A Framework for Building Robust and Trustworthy Machine Learning Models,” Comput. Fraud Secur., no. 5, pp. 65–74, 2024, doi: https://doi.org/10.52710/cfs.867.
7. J. Rahul Dattangire, Ruchika Vaidya, Divya Biradar, “Exploring the Tangible Impact of Artificial Intelligence and Machine Learning: Bridging the Gap between Hype and Reality,” 2024 1st Int. Conf. Adv. Comput. Emerg. Technol., pp. 1–6, 2024.
8. D. Patel, “AI-Enhanced Natural Language Processing for Improving Web Page Classification Accuracy,” vol. 4, no. 1, pp. 133–140, 2024, doi: 10.56472/25832646/JETA-V4I1P119.
9. S. Garg, “AI-Driven Innovations in Storage Quality Assurance and Manufacturing Optimization,” Int. J. Multidiscip. Res. Growth Eval., vol. 6, no. 2, pp. 1083–1087, 2025, doi: 10.54660/.IJMRGE.2025.6.2.1083-1087.

10. S. K. Tiwari, "Quality Assurance Strategies in Developing High-Performance Financial Technology Solutions," Int. J. data Sci. Mach. Learn., vol. 05, no. 01, pp. 323–335, Jun. 2025, doi: 10.55640/ijdsml-05-01-26.

11. P. Chandrashekar and M. Kari, "A Study on Artificial Intelligence in Software Engineering with Methodologies , Applications , and Effects on SDLC," TIJER Int. Res. J., vol. 11, no. 12, pp. 932–937, 2024.

12. V. Prajapati, "Advances in Software Development Life Cycle Models: Trends and Innovations for Modern Applications," JGREC, vol. 1, no. 4, 2025, doi: 10.5281/zenodo.15222129.

13. K. Shah, C. R. Katru, and S. Gami, "Redefining Software Quality Assurance: A Deep Dive into Automated Testing Strategies," Rev. Contemp. Sci. Acad. Stud., vol. 5, no. 3, Mar. 2025, doi: 10.55454/rcsas.5.03.2025.002.

14. S. K. Das, "Modernizing QA Frameworks for Omni-Based Retirement Systems : A Shift from Manual Mainframe Testing to Intelligent Automation .," vol. 06, no. 5, pp. 68–89, 2025.

15. K. Patel, S. B. Thakare, and N. K. Balaraman, "Resilient Hardware Lifecycle Management Through Agile Engineering Approaches for Sustainable Prototype-To-Production Transitions," 2026, pp. 292–307. doi: 10.1007/978-3-032-28616-1_23.

16. N. Medeshetty, A. N. Ghazi, S. Alawadi, and F. Alkhabbas, "From Requirements to Test Cases : An NLP-Based Approach for High-Performance ECU Test Case Automation," 2025, doi: 10.48550/arXiv.2505.00547.

17. Jariwala, "Construction Execution Methodologies for Modular Data Center Deployment: A Field-Based Framework for Mission-Critical Infrastructure Delivery," J. Inf. Syst. Eng. Manag., vol. 11, no. 3s, pp. 2630–2643, Sep. 2026.

18. Duggasani, "The Integration of Java-Based Applications in Software Development and AI: A Conceptual Review," Int. J. Adv. Res. Sci. Commun. Technol., vol. 5, no. 1, p. 11, Sep. 2025.

19. Bhanushali, "Ensuring Software Quality Through Effective Quality Assurance Testing: Best Practices and Case Studies," Int. J. Adv. Eng. Res., vol. 26, no. 4, pp. 1–18, 2023.

20. H. Sivaraman, "Natural Language Processing For Automated Test Case Generation From Software Requirement Documents," Int. J. Core Eng. Manag., vol. 5, no. 12, March, pp. 99–104, 2019.

21. S. R. Sagili, "Prompt-Instructed Generative based AI for Enhancing Transformer effectiveness Analysis," in 2024 Asian Conference on Intelligent Technologies (ACOIT), 2024, pp. 1–5. doi: 10.1109/ACOIT62457.2024.10939616.

22. J. W. Lim et al., "Test case information extraction from requirements specifications using NLP-based unified boilerplate approach," J. Syst. Softw., vol. 211, p. 112005, May 2024, doi: 10.1016/j.jss.2024.112005.

23. S. Kumar, K. Napte, R. Rani, and S. K. Pippal, "A method for IoT devices test case generation using language models," MethodsX, vol. 14, p. 103340, Jun. 2025, doi: 10.1016/j.mex.2025.103340.

24. H. P. Kapadia, "Voice and Conversational Interfaces in Banking Web Apps," J. Emerg. Technol. Innov. Res, vol. 8, no. 6, pp. g817--g823, 2021.

25. G. Maddali and S. J. Wawge, Site Reliability Engineering. 2025.

26. Dudhipala, R. Karne, and P. K. Pativada, "Prompt2Graph: Leveraging LLMs to Construct Knowledge Graphs from Technical Manuals," in 2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), IEEE, Sep. 2025, pp. 912–919. doi: 10.1109/ICIMIA67127.2025.11200177.

27. F. D. Ashkezari, "Investigating the usefulness of LLMs in Test Case Generation," 2024.

28. S. P. Kalava, "Enhancing Software Development with AI-Driven Code Reviews," North Am. J. Eng. Res., vol. 5, no. 2, pp. 1–7, 2024.

29. R. Ande, "A Unified Optimization Framework for Large Language Models in Enterprise Applications Using Python," J. Comput. Anal. Appl., vol. 33, no. 6, pp. 2111–2122, 2024.

30. G. Kathiresan, "Automated Test Case Generation with AI: A Novel Framework for Improving Software Quality and Coverage," World J. Adv. Res. Rev., vol. 23, no. 2, pp. 2880–2889, Aug. 2024, doi: 10.30574/wjarr.2024.23.2.2463.

31. M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda, and P. Abrahamsson, "A Tool for Test Case Scenarios Generation Using Large Language Models," 2024, doi: 10.48550/arXiv.2406.07021.

32. S. Thangavel, S. Srinivasan, S. B. V. Naga, and K. Narukulla, "Distributed Machine Learning for Big Data Analytics: Challenges, Architectures, and Optimizations," Int. J. Artif. Intell. Data Sci. Mach. Learn., vol. 4, no. 3, pp. 18–30, Oct. 2023, doi: 10.63282/3050-9262.IJAIDSML-V4I3P103.

33. R. Q. Majumder, "Machine Learning for Predictive Analytics: Trends and Future Directions," Int. J. Innov. Sci. Res. Technol., vol. 10, no. 4, pp. 3557–3564, 2025.

34. X. Zhao, H. Wang, J. Ding, Z. Hu, Q. Tian, and Y. Wang, "Augmenting software quality assurance with AI and automation using PyTest-BDD," Autom. Softw. Eng., vol. 33, no. 1, p. 23, May 2026, doi: 10.1007/s10515-025-00566-w.

35. N. Li, Y. Xu, J. Luo, and J. Fang, "AI-Driven Test Case Generation Based on DeepSeek-Chat Large-Language Model," in 2025 5th International Symposium on Computer Technology and Information Science (ISCTIS), IEEE, May 2025, pp. 1–4. doi: 10.1109/ISCTIS65944.2025.11065082.

36. Z. U. Abideen and G. Junxia, "Intent Based E2E Automated Test Case Generation for Web Applications Using LLM," in 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, Jul. 2025, pp. 1281–1290. doi: 10.1109/COMPSAC65507.2025.00161.

37. L. Naimi, E. M. Bouziane, M. Manaouch, and A. Jakimi, "A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering," in 2024 International Conference on Circuit, Systems and Communication (ICCSC), IEEE, Jun. 2024, pp. 1–5. doi: 10.1109/ICCSC62074.2024.10616548.

38. H. Edirisinghe and D. Wickramaarachchi, "Quality Assurance for LLM-Generated Test Cases: A Systematic Literature Review," in 2024 8th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI), IEEE, Dec. 2024, pp. 1–6. doi: 10.1109/SLAAI-ICAI63667.2024.10844968.

39. F. Chen, Y. Zhao, and Z. Li, "Research on Software Test Case Generation based on SysML," in 2023 International Symposium on Intelligent Robotics and Systems (ISoIRS), IEEE, May 2023, pp. 173–177. doi: 10.1109/ISoIRS59890.2023.00044.