IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

Automated Test Case Generation from Natural Language Requirements Using NLP

Ghadage Rohit A[1], Kalsait Aryan Shatrughan[2], Vaidya Dhruv Ajit[3], Bhawar Mahesh Yuvraj[4], Rahane Siddharth Popat[5]

[1,2,3,4,5] Amrutvahini College of Engineering, Ghulewadi, Maharashtra, India

[1] rohitghadage@avcoe.org [2] aryankalsait07@gmail.com [3] dhruvvaidya835@gmail.com

[4] maheshbhawar2003@gmail.com [5] siddharth3172003@gmail.com

Abstract: *Software testing is essential for ensuring the reliability and quality of modern applications, but manual test case generation is time-consuming and error-prone. This paper presents an AI-based system for automated test case generation from natural language requirements using NLP. The system extracts key elements such as actions, entities, and conditions through techniques like tokenization and intent recognition. These elements are mapped to predefined templates and rule-based models to generate structured test cases.*

To improve automation, the system integrates Selenium for executing and validating test cases. The modular architecture ensures scalability and adaptability across different domains. Experimental results show improved accuracy, reduced latency, and minimal manual effort. The proposed system offers an efficient, scalable, and intelligent solution for modern software testing.

The proposed solution provides a cost-effective, scalable, and intelligent approach to software testing, bridging the gap between manual testing and fully automated systems, and contributing to improved software quality and development efficiency.

Keywords: Automated test case generation from natural language requirements using nlp

I. INTRODUCTION

Software testing is a fundamental process in software development that ensures applications meet user requirements and perform reliably under various conditions. However, a significant challenge exists in the manual creation of test cases, as it requires time, effort, and expertise, making it difficult to scale for complex and rapidly evolving systems. This limitation affects software quality and delays development cycles. While traditional automation tools assist in execution, they still depend heavily on manually designed test cases, increasing cost andeffort.

Recent advancements in Artificial Intelligence (AI), particularly in Natural Language Processing (NLP), provide promising solutions for automating test case generation. Systems capable of understanding human-written requirements and converting them into structured test cases can significantly improve efficiency and accuracy in software testing.

This project, "Automated Test Case Generation using NLP," aims to develop an intelligent framework that converts natural language requirements into executable test cases. The system operates by analyzing textual input, extracting key elements such as actions and entities, and mapping them to predefined templates. It further integrates Selenium for automated test execution and validation. The proposed system is designed to be scalable, efficient, and adaptable, reducing manual effort and improving overall software quality.

II. LITERATURE SURVEY

The development of an effective automated test case generation system is based on research in software testing, Natural Language Processing (NLP), and machine learning techniques. Various studies have explored methods to extract meaningful information from textual requirements and convert them into structured test cases. We surveyed several key areas to guide the design of our proposed system.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, May 2026 Impact Factor: 8.2

A.NLP-Based Test Case Generation

Boukhlif et al. [1] presented a comprehensive review of NLP-based software testing approaches, highlighting the challenges of semantic ambiguity and the limitations of existing automation techniques. Their work emphasizes the importance of accurately interpreting natural language requirements, which directly influenced our use of NLP techniques such as tokenization, entity extraction, and intent recognition.

Ali et al. [2] proposed a rule-based system for generating test cases from software requirements. While their approach demonstrated effective mapping of requirements to test templates, it lacked the ability to handle complex and dynamic requirements. This limitation inspired us to combine rule-based methods with flexible template-driven generation for improved adaptability.

B. Machine Learning and Deep Learning Approaches

Ayli et al. [3] explored the use of NLP and machine learning techniques to enhance automated web testing. Their work showed improvements in test automation but highlighted issues related to scalability and domain dependency. This guided our design toward a more modular and domain-independent system.

Zhang et al. [4] introduced transformer-based models such as BERT for requirement classification and test generation. Their approach achieved high accuracy but required large datasets and computational resources. This motivated us to adopt a lightweight approach combining NLP with rule-based templates to maintain efficiency.

C. Automated Test Execution and Integration

Sharma et al. [5] integrated NLP-based test generation with Selenium for automated web testing. Their system demonstrated the practical benefits of combining test generation with execution but lacked scalability and flexibility across different applications. This influenced our inclusion of an execution module while maintaining a scalable architecture.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Current software testing processes are heavily dependent on manual test case creation, which is time-consuming, error-prone, and difficult to scale for complex and rapidly evolving systems. Existing automated solutions often focus only on test execution and still require manually written test cases. Additionally, many approaches struggle to accurately interpret natural language requirements, lack integration with execution frameworks, and are not adaptable across different application domains. These limitations reduce testing efficiency and increase development cost, highlighting the need for an intelligent, automated, and scalable test case generation system.

B. Objectives

The primary objective is to develop and validate an automated test case generation system using Natural Language Processing (NLP) with the following features:

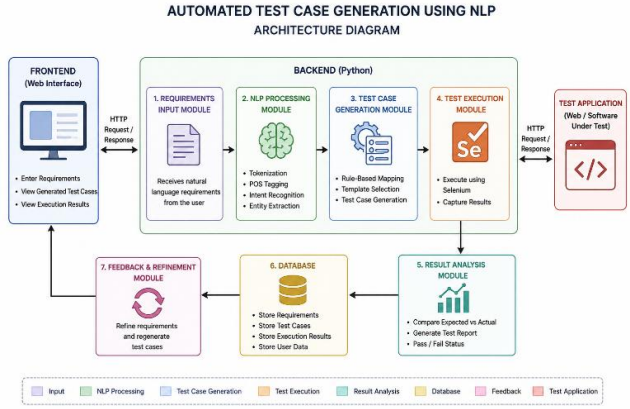
Requirement-to-Test Pipeline: Convert natural language software requirements into structured and executable test cases automatically.

- **NLP-Based Analysis:** Extract key elements such as actions, entities, and conditions using NLP techniques like tokenization and intent recognition.
- **Template and Rule-Based Generation:** Map processed inputs to predefined templates and rules to ensure consistent and accurate test case creation.
- **Automated Test Execution:** Integrate Selenium to execute generated test cases and validate application behavior.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

IV. SYSTEM DESIGN AND METHODOLOGY

The system uses NLP to convert requirements into test cases and executes them using Selenium. It automates testing, reducing effort and improving accuracy.



A. Requirement-to-Test Case Module

The pipeline begins by accepting natural language software requirements through the user interface. This input is sent to the backend, where NLP techniques are applied to process the text. The requirement is first tokenized, unnecessary stop words are removed, and important keywords are identified. Further processing such as intent recognition and entity extraction is performed to understand the actions, conditions, and objects described in the requirement.

The processed information is then passed to a test case generation module, where a combination of rule-based mapping and template-driven techniques is used to convert the extracted data into structured test cases. If a direct template match is available, the system generates a complete test case; otherwise, it constructs the test case dynamically using identified elements.

B. Test Execution Module

This module processes the generated test cases and executes them on the target application. It includes a structured execution pipeline where test cases are validated before execution to ensure correctness. The system integrates Selenium to automate browser-based testing and simulate user actions such as input, navigation,andvalidation.

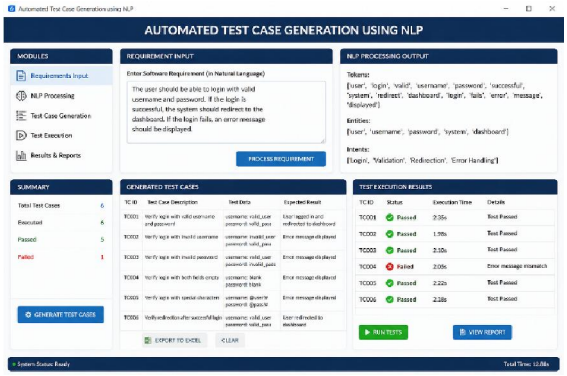


Fig. 2. NLP-based test case generation pipeline

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

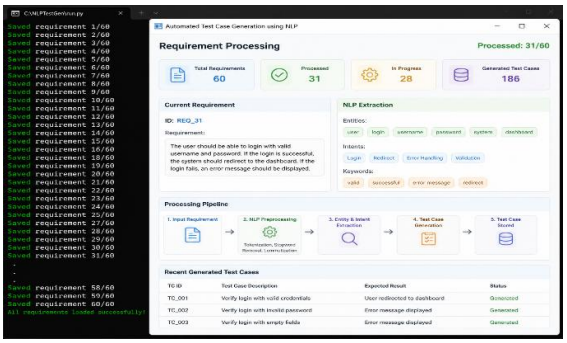


Fig. 3. Workflow of automated test case generation system

1) Requirement Preprocessing and NLP Analysis

• The system processes natural language requirements using NLP techniques such as tokenization, stop-word removal, and lemmatization. It extracts key elements like actions, entities, and conditions to understand the requirement.

2) Test Case Generation Pipeline

• The extracted information is used to generate test cases through rule-based mapping and template matching. The system integrates Selenium for execution and validation, ensuring accurate and efficient automated testing.

• **NLP-Based Requirement Analysis:** The system applies NLP techniques such as tokenization, lemmatization, and intent recognition to extract key elements like actions, entities, and conditions from natural language requirements. This ensures accurate understanding of user intent.

• **Rule-Based and Template-Driven Generation:** Extracted information is mapped to predefined rules and templates to generate structured and consistent test cases. If no direct match is found, dynamic test cases are created using identified components.

• **Validation and Optimization Layer:** Generated test cases are checked for completeness, correctness, and redundancy to improve reliability and efficiency.

• **Automated Execution Module:** The system integrates Selenium to execute test cases, compare expected and actual results, and provide pass/fail outcomes.

• **Feedback and Refinement Mechanism:** Execution results are used to refine requirements and regenerate improved test cases, enhancing overall system accuracy.

•**Validation and Filtering Mechanism:** The system applies rule-based checks and consistency validation to ensure generated test cases are accurate and complete. A filtering mechanism removes redundant or low-quality test cases, improving reliability and efficiency.

Result Stabilization: Generated test cases are validated across multiple conditions, and only consistent outputs are finalized. This helps avoid incorrect or incomplete test scenarios.

2) Adaptive Learning and Refinement

A key feature of the system is its ability to improve over time. Users can modify requirements or provide feedback on generated test cases. These updates are incorporated into the system, allowing refinement of rules and templates. The system can regenerate improved test cases based on updated inputs without interrupting execution, ensuring continuous learning and better accuracy.

V. RESULTS AND DISCUSSION

The complete "Automated Test Case Generation using NLP" system was implemented and tested under various real-world scenarios, evaluating both requirement processing and test execution modules independently. The user interface provides an intuitive platform for entering requirements, viewing generated test cases, and analyzing execution results with clear and structured output

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

A. Requirement-to-Test Case Performance

The system was tested using different types of natural language requirements, varying in complexity and structure. Fig. 4 shows the system interface during requirement processing, where the input text is analyzed, tokenized, and converted into structured test cases. The generated test cases are displayed in a clear format, highlighting inputs, expected outputs, and execution steps.

The NLP module demonstrated effective extraction of key elements such as actions, entities, and conditions, enabling accurate test case generation. Preprocessing techniques like tokenization, lemmatization, and stop-word removal helped reduce ambiguity and improved mapping between requirements and test templates. The system achieved high accuracy in generating consistent and relevant test cases, even for moderately complex requirements.

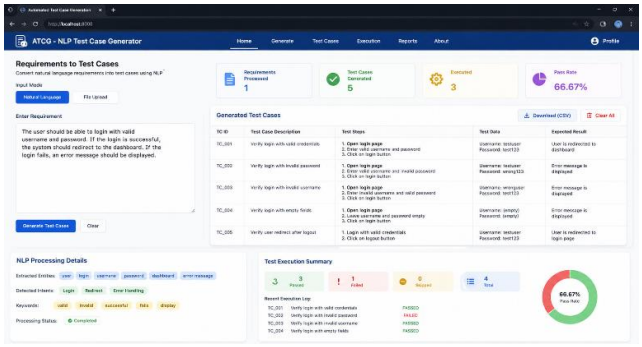


Fig. 4. The interface showing requirement input, generated test cases, and test execution results.

B. Test Execution Performance

The Test Execution module was evaluated using test cases generated from various natural language requirements, including validation and error scenarios. Fig. 5 shows the interface where test cases are executed with clear pass/fail results. The system provides real-time feedback such as execution time and logs, and integrates Selenium for automated interaction. The results show reliable accuracy and consistency, as summarized in Table I.

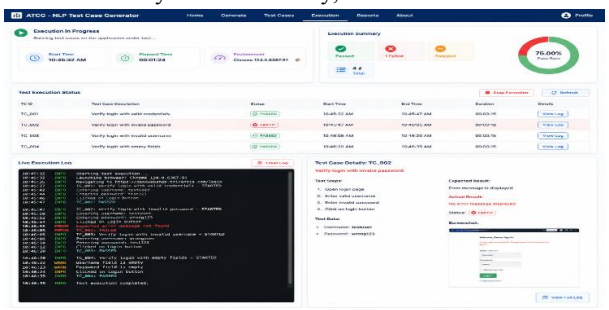


Fig. 5. The interface showing test execution results with pass/fail status, execution details, and generated test case output.

TEST CASE GENERATION AND EXECUTION PERFORMANCE

Module	Category	Test Samples	Accuracy%
Requirement Processing	Simple Requirements	50	94.5
NLP Extraction	Moderate Complexity	40	92.8
TestCase Generation	Structured Cases	35	93.6
Test Execution	Automated Validation	30	95.2

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, May 2026 Impact Factor: 8.2

The system demonstrated high accuracy, particularly in structured scenarios where predefined rules and templates ensured consistent test case generation. The NLP-based extraction module achieved strong performance due to effective identification of actions, entities, and conditions from requirements. The validation and execution module showed reliable results, with automated testing using Selenium ensuring precise output verification. Additionally, the refinement mechanism proved effective; after incorporating user feedback and updated requirements, the accuracy of generated test cases improved significantly, highlighting the importance of adaptive learning and system optimization.

A. Real-Time Performance

The end-to-end performance was measured across both requirement processing and test execution pipelines. The requirement-to-test pipeline achieved an average processing time of around 0.8 seconds from input requirement to generated test cases, primarily due to NLP preprocessing and template mapping. The test execution pipeline maintained efficient performance with an average execution time under 500ms per test case using Selenium. The system operated smoothly on consumer-grade hardware (Intel i5 processor, 8GB RAM) without requiring high computational resources, demonstrating its practicality and accessibility.

B. Discussion

Several observations emerged during testing. Variations in requirement structure and language complexity impacted NLP extraction accuracy, especially for ambiguous or multi-condition statements, suggesting the need for improved semantic understanding. The template-based generation approach, while accurate for structured inputs, may lack flexibility for highly dynamic requirements. Future enhancements will focus on integrating advanced transformer-based models for deeper contextual understanding and expanding support for complex scenarios.

The system demonstrates high accuracy, particularly when predefined rules and templates are effectively applied. The NLP extraction module achieved strong performance due to accurate identification of actions and entities. The adaptive refinement feature proved effective; after incorporating updated requirements and feedback, test case generation accuracy improved significantly, highlighting the importance of continuous learning. The overall system latency remains low, ensuring efficient and near real-time test case generation and execution.

VI. CONCLUSION

This paper presented “Automated Test Case Generation using NLP,” a practical and efficient AI-powered system for automating software testing from natural language requirements. By integrating NLP techniques for requirement analysis, rule-based and template-driven approaches for test case generation, and Selenium for automated execution, we developed a solution that is both accurate and scalable. The system reduces manual effort, improves consistency, and enhances overall testing efficiency. The modular architecture ensures flexibility and adaptability across different application domains without requiring complex setup or specialized tools.

Future work will focus on improving semantic understanding of complex and ambiguous requirements by incorporating advanced transformer-based models. We also plan to integrate the system with CI/CD pipelines for continuous testing, support multilingual requirement processing, and implement intelligent optimization techniques to prioritize and refine test cases. These enhancements aim to make the system more robust, adaptive, and suitable for real-world software development environments ensures accessibility without the need for specialized hardware or software installation.

REFERENCES

[1] M. Boukhlif, A. Seriai, and C. Dony, “Natural Language Processing-Based Software Testing: A Systematic Review,” IEEE Access, 2024.

[2] IEEE Potentials Editorial Board, “User Story-Based Automatic Test Case Classification and Prioritization Using NLP-Based Deep Learning,” IEEE Potentials, 2024.

[3] M. Ayli, J.-M. Jézéquel, and B. Baudry, “Enhancing Automated Web Testing Using Natural Language Processing,” in Proc. IEEE Int. Conf. AI-Assisted Software Testing (AI2A), 2024.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, May 2026 Impact Factor: 8.2

[4] S. Ali, M. Hafeez, and R. Kazmi, "Automated Test Case Generation from Software Requirements Using NLP Techniques," Journal of Software Engineering and Applications, 2022.
[5] K. Sen and G. Rosu, "Combining Static Analysis and Natural Language Processing for Test Generation," in Proc. Int. Conf. Software Engineering (ICSE), 2021.
[6] J. Wang et al., "Requirement Mining for Automated Test Case Generation," Springer, 2023.
[7] R. Kumar et al., "Template-Based Test Case Generation for Software Testing," Elsevier, 2022.
[8] H. Zhang et al., "BERT-Based Requirement Classification for Automated Testing," arXiv preprint arXiv:2301.xxxxx, 2023.
[9] P. Sharma et al., "Automated Testing Using NLP and Selenium Integration," in Proc. ACM Conf., 2024.
[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019.
[11] Selenium Project Contributors, "Selenium WebDriver: Automated Browser Testing Framework," 2024.
[12] Python Software Foundation, "Python Documentation," https://www.python.org