iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 14 **Issue:** III **Month of publication:** March 2026

**DOI:** https://doi.org/10.22214/ijraset.2026.78324

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue III Mar 2026- Available at www.ijraset.com*

# Automatic Dependency Conflict Detection Tool for Software Projects

Ms. Savitha R[1], Ms. Udhayarani K S[2], Mrs. R Sindhiya[3]
*Department of Computer Science and Engineering KLN College of Engineering, Anna University, Madurai, India*

***Abstract:*** ***Software projects today depend on numerous external libraries and packages to provide additional functionality and improve development efficiency. However, managing these dependencies in large software systems can become complex and often leads to dependency conflicts, especially when different components require incompatible versions of the same library. These conflicts may cause system failures, security vulnerabilities, and maintenance difficulties. Therefore, detecting and resolving dependency conflicts early in the development process is an important challenge in modern software engineering. This paper presents an Automatic Dependency Conflict Detection Tool for Large Software Projects that analyzes dependency files and identifies potential conflicts in software environments. The system uses a dependency parsing mechanism to extract dependency information from project files and applies rule-based analysis to detect direct and transitive dependency conflicts. Additionally, the system provides suggestions for resolving detected conflicts and predicts potential risk levels associated with them. A web-based interface is also developed to visualize the analysis results and provide developers with an easy way to understand dependency relationships. The proposed system improves dependency management by providing automated detection, conflict analysis, and visualization of dependency structures. Experimental results demonstrate that the tool effectively identifies dependency conflicts and assists developers in resolving them quickly. This approach enhances software stability and reduces development time in large-scale software projects. Future enhancements may include integrating machine learning techniques for more intelligent conflict prediction and automated dependency optimization.***

***Keywords: Dependency Analysis, Conflict Detection, Software Engineering, Dependency Management, Risk Prediction.***

## I. INTRODUCTION

Modern software development relies heavily on external libraries and third-party packages to accelerate development and provide additional functionality. These libraries are managed through dependency management systems that allow developers to integrate multiple components into a single application. While this approach improves productivity, it also introduces challenges related to dependency compatibility. In large software projects, different modules may require different versions of the same library, which can lead to dependency conflicts and system instability.

Dependency conflicts occur when multiple packages require incompatible versions of the same dependency. Such conflicts may result in build failures, runtime errors, and security vulnerabilities. As software systems grow in size and complexity, manually identifying and resolving these conflicts becomes increasingly difficult for developers. Existing dependency management tools provide basic support for version management, but they often fail to detect complex transitive dependencies and hidden conflicts across multiple components.

In dependency analysis, the total number of conflicts in a project can be estimated based on the dependencies detected in the system. A simple representation for calculating the total number of conflicts is given by:

$$C = \sum_{i=1}^{n} D_i$$

where:

- **C** represents the total number of detected dependency conflicts
- $\mathbf{D_i}$ represents the conflicts associated with the dependency
- **n** represents the total number of dependencies in the project

This formula provides a basic representation for analysing dependency conflicts across multiple libraries in large software projects.

To address these challenges, this paper proposes an Automatic Dependency Conflict Detection Tool for Large Software Projects. The proposed system automatically analyses dependency files, extracts dependency relationships using a dependency parsing mechanism, and identifies both direct and transitive dependency conflicts. In addition, the system provides conflict resolution suggestions and predicts potential risk levels associated with detected conflicts. A web-based visualization dashboard is also implemented to help developers understand dependency structures and conflict patterns more effectively.

The main objective of this research is to improve dependency management in large software environments by providing automated conflict detection and analysis. The proposed approach helps developers identify problems early in the development process, reducing maintenance effort and improving software reliability.

## II. MOTIVATION OF THE STUDY

The motivation for this study arises from the increasing complexity of modern software projects that rely on numerous external libraries and dependencies. Managing these dependencies manually becomes difficult as projects grow larger, often leading to version conflicts, system instability, and increased debugging time. Developers may spend significant effort identifying and resolving these conflicts, which can slow down the development process. Therefore, there is a need for an automated system that can analyze dependency files, detect potential conflicts early, and provide useful insights to developers. The proposed dependency conflict detection tool aims to simplify dependency management and improve the reliability and stability of large software projects.

## III. PROPOSED SYSTEM

The proposed system presents an Automatic Dependency Conflict Detection Tool for Large Software Projects designed to identify and analyse dependency conflicts in software development environments. Modern software applications depend on multiple external libraries and packages, which often lead to compatibility issues when different modules require different versions of the same dependency. The proposed system aims to automatically detect these conflicts and assist developers in resolving them efficiently.

The system begins by accepting dependency files such as requirements.txt, package.json, or similar configuration files from software projects. A dependency parser extracts the list of libraries and their respective versions from these files. After extracting the dependency information, the system analyses the relationships between libraries and identifies both direct dependencies and transitive dependencies. This analysis helps in understanding the complete dependency structure of the software project.

Once the dependencies are parsed, a conflict detection engine examines the versions of the libraries and identifies potential incompatibilities. A conflict occurs when multiple modules require different versions of the same dependency.

After detecting conflicts, the system also provides recommended solutions to developers, such as selecting compatible versions or updating dependent libraries. Additionally, the system includes a risk prediction module that estimates the severity of conflicts based on the number of dependencies and conflicts detected. The analysis results are displayed through a web-based dashboard, allowing developers to visualize dependency relationships, detected conflicts, and suggested resolutions.

The proposed system improves the dependency management process by automating conflict detection and providing clear insights into dependency structures. This helps developers reduce debugging time, improve software stability, and maintain large-scale software projects more efficiently.

## IV. SYSTEM ARCHITECTURE

The system architecture of the Automatic Dependency Conflict Detection Tool for Large Software Projects consists of several interconnected modules that work together to analyse software dependency files and identify potential conflicts. The process begins with the Input Module, where users upload dependency files such as requirements.txt or package.json. These files are processed by the Dependency Parsing Module, which extracts library names, versions, and relationships between packages. The extracted data is then analysed by the Conflict Detection Engine, which identifies direct and transitive dependency conflicts by comparing version requirements.

After detecting conflicts, the Risk Prediction Module evaluates the severity of conflicts based on the number of dependencies involved, version incompatibilities, and potential impact on the overall project stability. This module assigns a risk score that helps developers understand the seriousness of the detected issues.

The analysed data is then passed to the Recommendation Module, which suggests possible solutions such as upgrading, downgrading, or replacing conflicting dependencies to maintain compatibility within the project environment. Finally, the results are displayed through a Visualization Dashboard, allowing developers to easily view dependency structures, detected conflicts, risk levels, and recommended solutions for resolving them.

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue III Mar 2026- Available at www.ijraset.com*

This modular architecture ensures scalability, maintainability, and efficient analysis of dependency structures in large-scale software development environments.

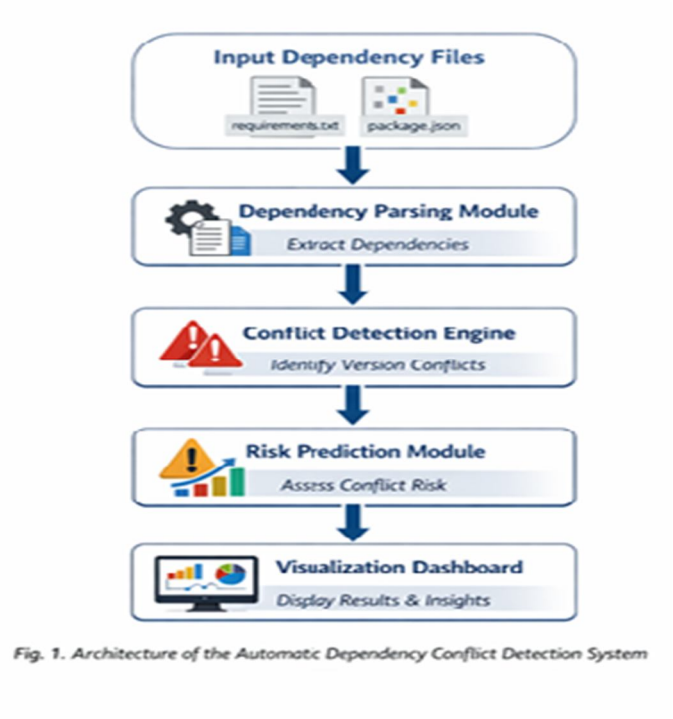



*Fig. 1. Architecture of the Automatic Dependency Conflict Detection System*

## V. IMPLEMENTATION

The implementation of the Automatic Dependency Conflict Detection Tool for Large Software Projects is carried out using modern web technologies and programming frameworks to analyse dependency files and detect conflicts efficiently. The system is developed using Python as the primary programming language due to its strong support for data processing and software analysis tasks. A Flask web framework is used to create a lightweight web-based interface that allows users to upload dependency files and view the analysis results.

The system processes dependency files such as requirements.txt and package configuration files, which contain the list of libraries and their respective versions used in the project. The dependency parsing module extracts these libraries and organizes them into structured data for analysis. After extracting the dependency information, the conflict detection module compares the versions of different libraries to identify direct and transitive conflicts.

To present the analysis results clearly, the system uses HTML templates and JavaScript libraries such as Chart.js for visualization. These tools help display graphical representations of dependencies, detected conflicts, and risk levels in an interactive dashboard. The implementation also includes a risk prediction mechanism that evaluates the severity of conflicts based on the number of dependencies and detected incompatibilities.

Overall, the implemented system provides an efficient way for developers to upload dependency files, analyse dependency structures, detect potential conflicts, and visualize the results through an easy-to-use web interface.

## VI. RESULTS AND DISCUSSION

The proposed Automatic Dependency Conflict Detection Tool for Large Software Projects was tested using several sample dependency files obtained from different software projects. The system successfully parsed the uploaded dependency files and extracted information about libraries, versions, and dependency relationships. The dependency parsing module was able to identify both direct and transitive dependencies, allowing the system to build a complete dependency structure for analysis.

During the experiment, the tool analysed multiple dependency files and detected several version incompatibilities between libraries. The conflict detection engine identified cases where different modules required different versions of the same library. These conflicts were automatically highlighted, and the system provided suggested solutions such as selecting compatible versions or updating dependent libraries. The visualization dashboard displayed the results in a clear format, including the total number of dependencies, detected conflicts, and recommended resolutions.

The results show that the system effectively identifies dependency conflicts and helps developers understand complex dependency relationships within large software projects. The risk prediction module also provided useful insights by estimating the severity of conflicts based on the ratio of conflicts to total dependencies. This analysis helps developers prioritize critical conflicts that may affect software stability. Overall, the proposed system improves dependency management by automating conflict detection and providing clear visualization of the analysis results, thereby reducing manual effort and improving software reliability.


Fig 1. Website

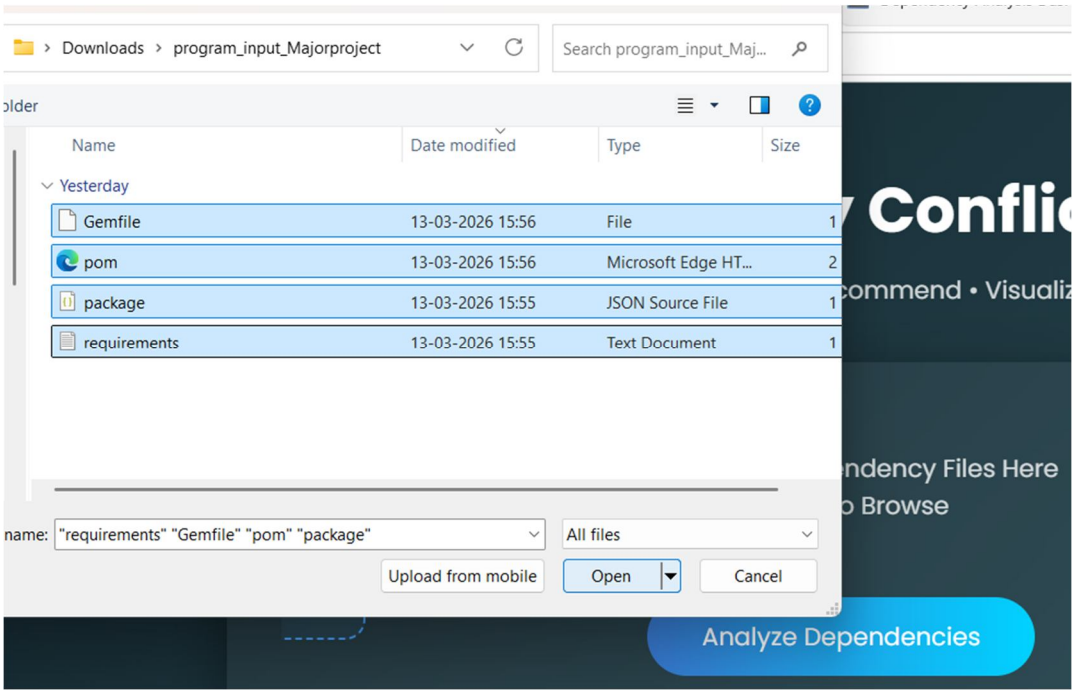



Fig 2. Selecting Dependency files

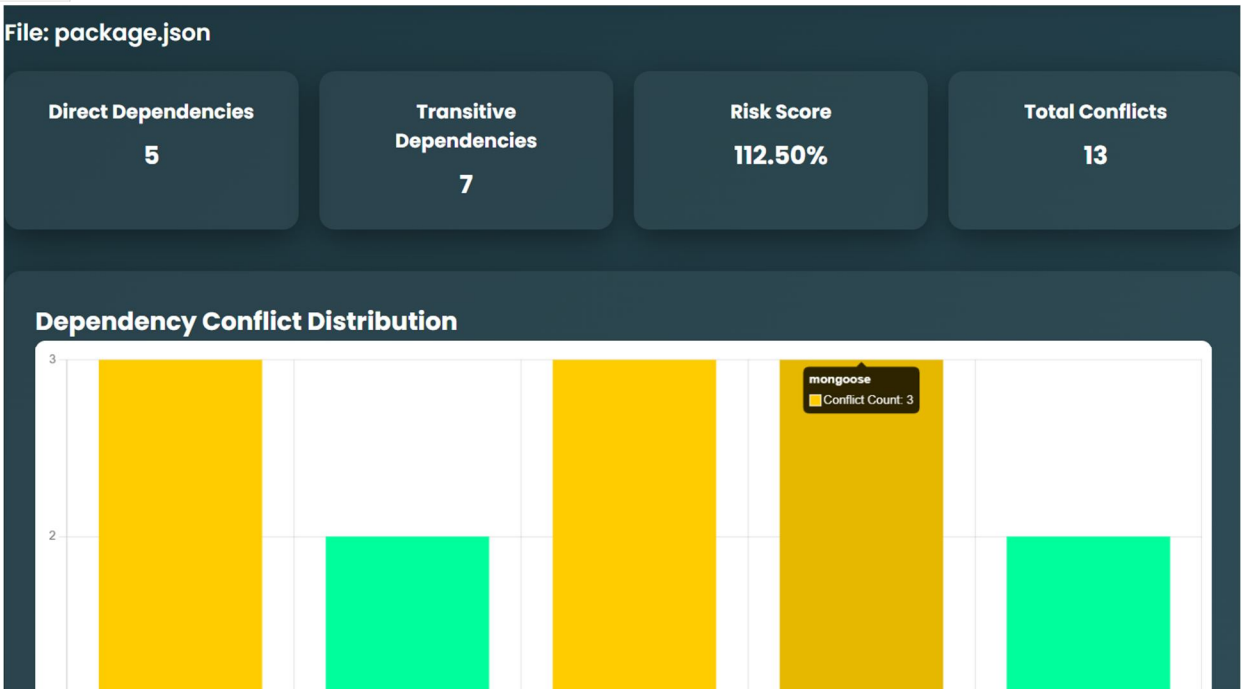



Fig 3. Dependency Conflict

**Package Conflict Details**

| Package | Conflict Count | Severity | Recommended Version |
|---|---|---|---|
| express | 2 | Low | 5.0.0 |
| react | 2 | Low | 18.0.0 |
| lodash | 3 | Medium | ^4.17.0 |
| axios | 3 | Medium | ^1.0.0 |
| mongoose | 3 | Medium | ^6.0.0 |

Fig 4. Package Conflict

## VII. CONCLUSION

In this paper, an Automatic Dependency Conflict Detection Tool for Large Software Projects has been proposed to address the challenges associated with managing dependencies in modern software development. Large software systems often rely on multiple external libraries, which may lead to compatibility issues when different modules require conflicting versions of the same dependency. The proposed system analyses dependency files, extracts library information through a dependency parsing mechanism, and automatically detects both direct and transitive dependency conflicts.

The implemented tool provides developers with a clear understanding of dependency relationships and highlights potential conflicts that may affect software stability. In addition, the system offers suggested solutions for resolving conflicts and includes a risk prediction mechanism that evaluates the severity of detected conflicts. The visualization dashboard presents the analysis results in an easy-to-understand format, helping developers quickly identify and address dependency issues.

The results demonstrate that the proposed system can effectively detect dependency conflicts and assist developers in improving dependency management in large software projects. By automating the conflict detection process and providing meaningful insights, the system helps reduce debugging time and improves overall software reliability.

## VIII. FUTURE WORK / ENHANCEMENTS

Although the proposed Automatic Dependency Conflict Detection Tool for Large Software Projects provides effective conflict detection and analysis, several improvements can be made in the future to enhance its capabilities. One possible enhancement is the integration of machine learning techniques to predict potential dependency conflicts more accurately by analysing historical project data and dependency patterns. This would allow the system to provide smarter recommendations for resolving compatibility issues.

Another future improvement is to support multiple dependency file formats from various programming ecosystems such as *Maven (pom.xml)*, *Gradle*, and *Node.js package files*. This would make the tool more flexible and useful for developers working with different technologies. In addition, the system can be extended to include real-time monitoring of dependencies and automatic updates to ensure that projects always use compatible and secure library versions.

Furthermore, the system could be integrated with continuous integration and continuous deployment (CI/CD) pipelines such as GitHub Actions or Jenkins. This would allow dependency conflicts to be detected automatically during the development and build process.

The system can also be improved by incorporating advanced visualization techniques to better represent complex dependency relationships. Interactive graphs and network-based visualizations can help developers easily understand dependency structures and identify conflict points within large projects. Such visual tools would improve the usability of the system and allow developers to analyse dependency networks more efficiently.

Furthermore, the tool can be integrated with continuous integration and continuous deployment (CI/CD) pipelines such as GitHub Actions, Jenkins, or GitLab CI. This integration would allow automatic dependency conflict detection during the software development lifecycle, ensuring that conflicts are identified early during the build or deployment stages.

Future enhancements may also include automated conflict resolution mechanisms, where the system not only detects conflicts but also suggests optimized dependency versions based on compatibility analysis. Additionally, incorporating security vulnerability detection within dependencies can further improve the reliability and safety of software systems. With these improvements, the proposed tool can evolve into a comprehensive platform for intelligent dependency management in large-scale software development environments.

## IX. ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project guide and faculty members for their valuable guidance, encouragement, and continuous support throughout the development of this project. Their suggestions and insights helped us improve our work and complete the project successfully.

We also thank the management and the Department of Computer Science and Engineering of our institution for providing us with the necessary facilities, resources, and a supportive environment to carry out this research work.

We extend our appreciation to our friends and team members who contributed ideas, discussions, and assistance during the development and testing phases of this project.

Finally, we would like to thank our families for their constant encouragement, motivation, and support throughout this journey.

## REFERENCES

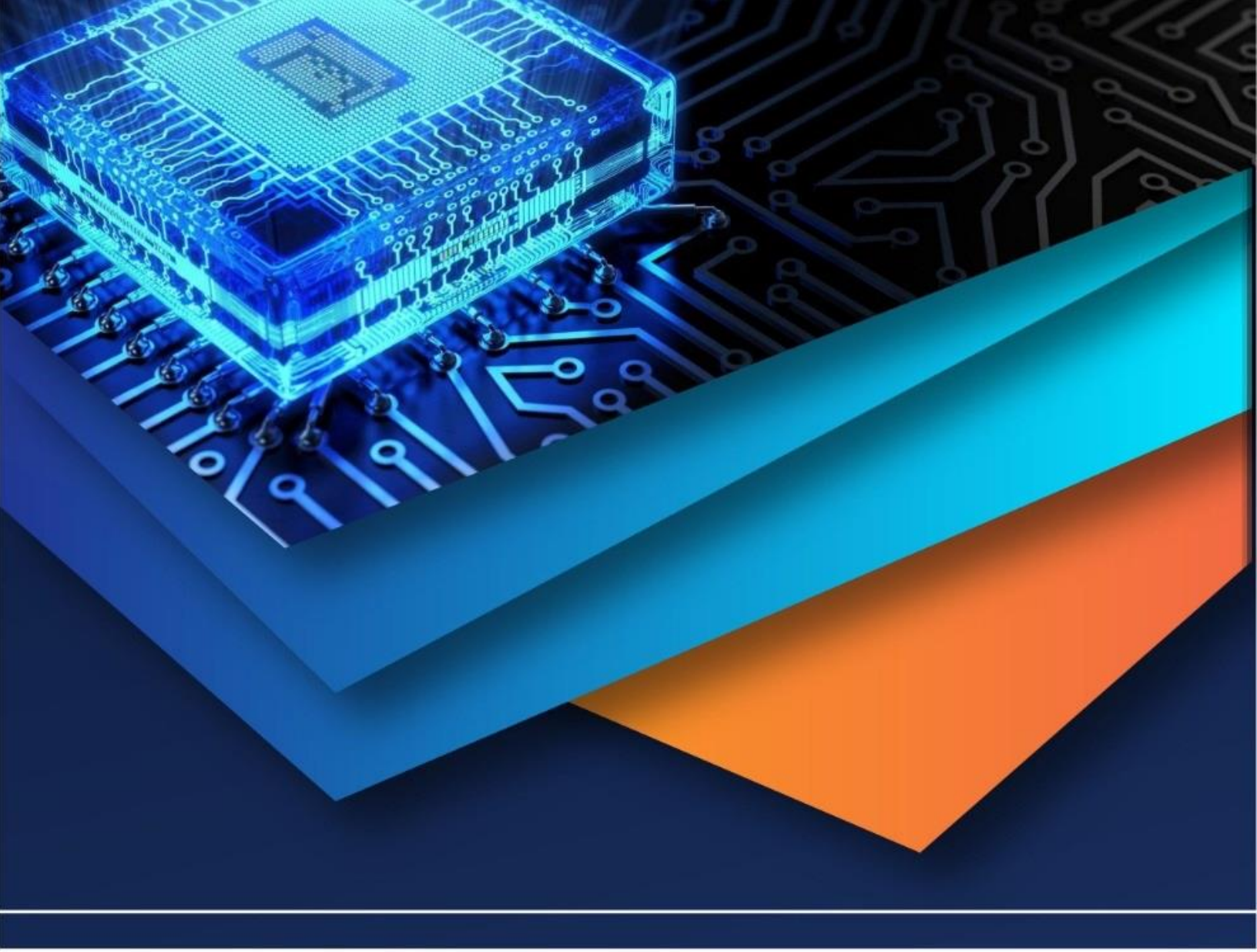
[1] M. Raatikainen, Q. Motger, C. M. Lüders, X. Franch, L. Myllyaho, E. Kettunen, J. Marco, J. Tiihonen, M. Halonen, and T. Männistö, “Improved Management of Issue Dependencies in Issue Trackers of Large Collaborative Projects,” *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2128–2148, Apr. 2023.

[2] H. Xia, J. Gluck, S. Simsek, D. S. Medina, and D. Starobinski, “VDGraph: A Graph-Theoretic Approach to Unlock Insights from SBOM and SCA Data,” arXiv preprint, Jul. 2025. [Online]. Available: https://arxiv.org/abs/2507.20502

[3] Z. Yu, R. Kumar, and J. Smith, “Reducing Alert Fatigue via AI-Assisted Negotiation: A Case for Dependabot,” arXiv preprint, Feb. 2025. [Online]. Available: https://arxiv.org/abs/2502.06175

[4] Y. Chen, X. Li, and P. Wang, “Out of Sight, Still at Risk: The Lifecycle of Transitive Vulnerabilities in Maven,” arXiv preprint, Apr. 2025. [Online]. Available: https://arxiv.org/abs/2504.04803

[5] P. Cheng, Y. Luo, and H. Zhang, “ReachCheck: Compositional Library-Aware Call Graph Reachability Analysis,” ACM Transactions on Software Engineering and Methodology (TOSEM), 2025. [Online]. Available:https://chengpengwang.github.io/publications/TOSEM2025.pdf



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY