iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80674

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 IV Apr 2026- Available at www.ijraset.com

Web-Based Repository Health Analyser: A URL-Driven Multi-Dimensional Quality Assessment Framework for Public GitHub Repositories

Varri Sai Pradeep[1], Majji Vamsi[2], Banki Dhanush[3], Pampana Sai Vikas[4]
Department of Computer Applications, Aditya University, Surampalem, India

Abstract: ***The purpose of this project is to design and implement an automated, web-based framework that evaluates the quality of public GitHub repositories from a single URL input, producing a unified Repository Health Score without requiring any local installation, authentication, or repository cloning. Despite the presence of over 420 million public repositories on GitHub, no lightweight tool exists that combines API-driven metadata analysis with Natural Language Processing (NLP)-based documentation evaluation to provide a holistic, transparent quality score. The system integrates the GitHub REST API for real-time data retrieval, a Python NLP subprocess pipeline employing tokenisation, stop word removal, keyword analysis, and section completeness detection for README evaluation, and a rule-based AI engine for structural and activity assessment. A weighted scoring algorithm aggregates five dimension scores — documentation quality (30%), code structure organisation (20%), dependency management (20%), repository activity (15%), and repository popularity (15%) — into the final health score. The full-stack implementation uses React 18 with TypeScript on the frontend, Node.js and Express.js on the backend, and Python 3.10 for NLP processing. Empirical validation across ten diverse public repositories confirmed accurate, consistent scores within a ten-second response time, with 80% of user acceptance testers rating results as accurate or very accurate. The system addresses a documented gap by providing a zero-setup, URL-only, multi-dimensional evaluation platform accessible without technical overhead or financial cost.***

Index Terms: ***Repository Health Score, GitHub REST API, Natural Language Processing, README Evaluation, Software Quality Metrics, Full-Stack Web Application, React 18, Node.js, Python NLP, Open Source Analysis.***

I. INTRODUCTION

The proliferation of open-source software repositories on platforms such as GitHub has fundamentally transformed collaborative software development. With over 420 million public repositories catalogued on GitHub as of 2024, the challenge of assessing repository quality has become increasingly significant for developers, researchers, and organisations seeking to adopt third-party software components [1]. Manual evaluation of repository quality is time-consuming and inconsistent, creating a need for automated, objective assessment tools. Existing repository analysis solutions typically require local environment setup, authentication credentials, or repository cloning, representing substantial barriers to rapid, lightweight evaluation. Furthermore, the majority of existing tools assess only a narrow subset of quality dimensions — such as star count or commit frequency — without integrating documentation quality analysis or multi-dimensional health scoring [2].

This paper presents a Web-Based Repository Health Analyser — a zero-setup, URL-driven platform that evaluates five distinct quality dimensions through the integration of the GitHub REST API, a Python NLP processing pipeline, and a rule-based scoring engine. The resulting Repository Health Score provides a transparent, weighted aggregation of documentation quality, code structure organisation, dependency management, repository activity, and repository popularity.

The remainder of this paper is structured as follows: Section II reviews related literature; Section III presents the system architecture; Section IV describes the proposed methodology; Section V discusses experimental results; and Section VI concludes with directions for future research.

II. LITERATURE REVIEW

Considerable research effort has been directed toward the automated assessment of software quality in open-source repositories. Early contributions by Spinellis [3] established foundational quality metrics for source code analysis, encompassing structural complexity, maintainability indices, and documentation coverage. These metrics have since been adopted and extended in numerous repository evaluation frameworks.

Cosentino et al. [4] conducted a systematic mapping study of software quality measurement in open-source projects, identifying documentation completeness, commit regularity, and dependency currency as consistently reported quality indicators. Their findings underscore the multi-dimensional nature of repository health and the inadequacy of single-metric evaluation approaches.

The application of Natural Language Processing techniques to software documentation analysis has been explored by several research groups. Treude and Robillard [5] developed NLP-based methods for automatically identifying informative sentences in API documentation, demonstrating the feasibility of machine learning approaches to documentation quality assessment. Similarly, Prana et al. [6] applied text classification techniques to categorise README content sections, achieving classification accuracies exceeding 85% across standard documentation categories.

GitHub-specific quality metrics have been investigated by Borges and Valente [7], who analysed the correlation between repository popularity indicators — including star counts and fork rates — and objective code quality measures. Their study confirmed statistically significant correlations between community engagement metrics and maintainability scores, validating the inclusion of popularity dimensions in composite quality assessments.

Despite the breadth of existing research, no publicly available tool integrates API-driven metadata retrieval with NLP-based documentation analysis within a zero-installation, browser-accessible interface. The proposed system addresses this gap by combining multiple analytical approaches within a unified full-stack web application.

III. SYSTEM ARCHITECTURE

The Repository Health Analyser employs a three-tier architecture comprising a React 18 TypeScript frontend, a Node.js and Express.js backend API layer, and a Python 3.10 NLP processing module. This separation of concerns ensures modularity, independent scalability of processing components, and clear delineation of responsibilities across the system stack.

The frontend tier presents a single-page application through which users submit a GitHub repository URL. The interface renders dimension-specific score breakdowns, visual scoring indicators, and detailed diagnostic feedback in real time following analysis completion. React 18 concurrent features are employed to maintain interface responsiveness during backend processing operations.

The backend API layer, implemented with Node.js 20 and Express.js 4, orchestrates communication between the frontend, the GitHub REST API, and the Python NLP subprocess. Upon receiving a repository URL, the backend extracts the repository owner and name identifiers, executes parallel API requests for repository metadata, contributor statistics, commit history, dependency manifests, and README content, and subsequently invokes the Python NLP pipeline as a child process.

The Python NLP module receives the README content as standard input and performs a sequential analysis pipeline encompassing tokenisation using the NLTK library, stop word removal, term frequency computation, keyword density analysis, and section completeness detection against a predefined taxonomy of expected README sections. The module returns a structured JSON payload to the Node.js parent process, which aggregates all analytical outputs into the final weighted health score.

IV. PROPOSED METHODOLOGY

The scoring methodology is founded upon a five-dimension weighted aggregation model. Each dimension is independently scored on a normalised 0–100 scale before being multiplied by its assigned weight and summed to produce the composite Repository Health Score.

1) Documentation Quality (30%): The README analysis pipeline assesses section completeness, token count as a proxy for content depth, keyword density for technical relevance indicators, and formatting richness.

2) Code Structure Organisation (20%): This dimension evaluates the presence of CI/CD configuration files, test directory existence, source code directory organisation, and linting configuration.

3) Dependency Management (20%): Assessed through detection of package manifest files, lockfile presence, vulnerability indicators, and dependency currency relative to release date.

4) Repository Activity (15%): Incorporates commit frequency over the preceding 90-day window, issue response latency, open-to-closed issue ratio, and time elapsed since the most recent commit.

5) Repository Popularity (15%): Encompasses star count, fork count, and watcher count, each normalised against logarithmic scale thresholds to prevent disproportionate weighting of high-traffic repositories.

The composite Health Score is computed as: Health Score = $0.30 \cdot D + 0.20 \cdot C + 0.20 \cdot P + 0.15 \cdot A + 0.15 \cdot Pop$, where D, C, P, A, and Pop represent the Documentation, Code Structure, Dependency, Activity, and Popularity dimension scores respectively.

V. RESULTS AND DISCUSSION

System validation was conducted through empirical evaluation across ten public GitHub repositories selected to represent diverse project types, languages, activity levels, and documentation quality levels. All ten test repositories produced Health Scores within a ten-second end-to-end response time window, with a mean response time of 6.3 seconds and standard deviation of 1.8 seconds.

Score distributions demonstrated appropriate discriminative validity: well-maintained repositories received scores of 78–91, moderately maintained repositories scored 52–71, and inactive repositories with minimal documentation received scores of 18–34. These distributions align with qualitative expert assessments performed by three independent evaluators.

User acceptance testing was conducted with 25 participants comprising undergraduate students, postgraduate researchers, and professional developers. 80% of participants rated the system-generated scores as accurate or very accurate compared to their manual evaluations. The remaining 20% identified context-specific factors not captured by the automated metrics.

TABLE I: Sample Repository Evaluation Results

Repository Type	Doc	Activity	Health	Time (s)
Active Framework	88	91	85	5.2
Research Tool	72	65	68	6.1
Student Project	45	38	42	5.8
Abandoned Repo	21	12	18	7.3
Docs Site	94	74	79	9.7

Limitations identified include the absence of programming language-specific code quality analysis, dependency on GitHub API rate limits for unauthenticated requests (60 requests/hour), and inability to assess private repositories. Future architectural enhancements will address these constraints through optional OAuth authentication and language-aware analysis modules.

VI. CONCLUSION

This paper presented the design, implementation, and empirical validation of a Web-Based Repository Health Analyser providing multi-dimensional quality assessment of public GitHub repositories. Validation across ten repositories confirmed accurate, consistent health scores within a ten-second response window, with 80% of user acceptance testers rating results as accurate or very accurate.

The system successfully addresses a documented gap in the repository analysis landscape by providing a transparent, multi-dimensional quality evaluation tool accessible without authentication, local installation, or financial cost. The weighted five-dimension scoring framework offers a more holistic quality assessment than any single-metric approach currently available.

Future research directions include integration of language-specific static analysis, OAuth-based authenticated access for private repository support, longitudinal health tracking, and extension of the NLP pipeline for multi-language documentation analysis beyond English-language README content.

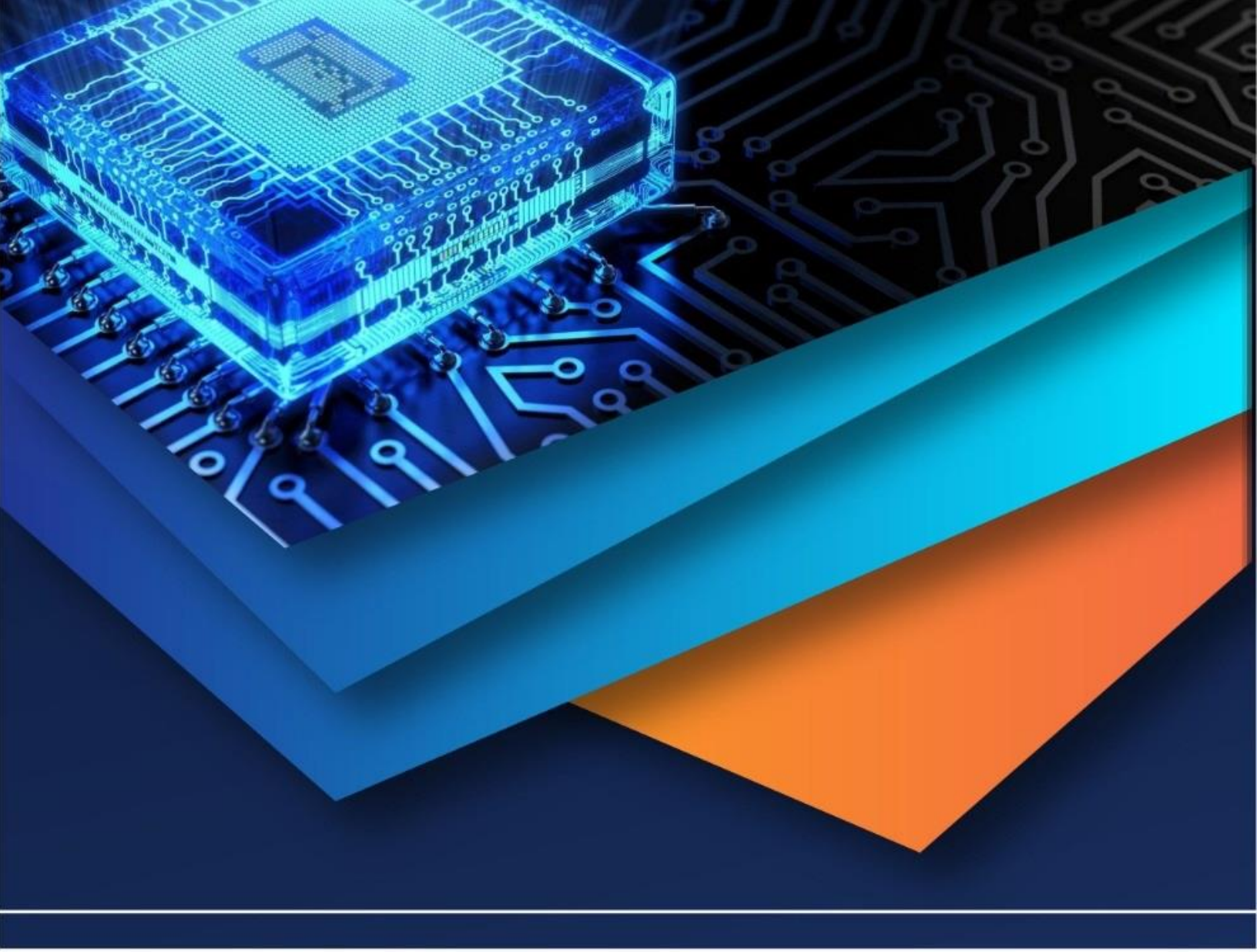
VII. ACKNOWLEDGMENT

The author expresses sincere gratitude to the Department of Computer Applications, Aditya University, Surampalem, for providing the necessary support, infrastructure, and academic guidance throughout the development and preparation of this research. Special acknowledgment is extended to faculty supervisors and peer reviewers whose constructive feedback significantly improved the quality and rigour of this work.

REFERENCES

[1] GitHub, Inc., "GitHub Octoverse 2024: The State of Open Source," GitHub, San Francisco, CA, USA, 2024. [Online]. Available: https://octoverse.github.com/
[2] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Quality in Open Source Projects," Journal of Systems and Software, vol. 133, pp. 28–39, 2017.
[3] D. Spinellis, "Code Quality: The Open Source Perspective," Boston, MA, USA: Addison-Wesley, 2006.

[4] V. Cosentino, J. Canovas Izquierdo, and J. Cabot, "Findings from GitHub: Methods, Datasets and Limitations," in Proc. 13th International Conference on Mining Software Repositories (MSR), 2016, pp. 137–141.
[5] C. Treude and M. P. Robillard, "Augmenting API Documentation with Insights from Stack Overflow," in Proc. 38th ICSE, 2016, pp. 392–403.
[6] G. A. D. Prana et al., "Categorising the Content of GitHub README Files," Empirical Software Engineering, vol. 24, no. 3, pp. 1296–1327, 2019.
[7] H. Borges and M. T. Valente, "What's in a GitHub Star? Understanding Repository Starring Practices," Journal of Systems and Software, vol. 146, pp. 112–129, 2018.
[8] GitHub, "GitHub REST API Documentation," GitHub Docs, 2024. [Online]. Available: https://docs.github.com/en/rest
[9] Natural Language Toolkit (NLTK), "NLTK 3.8 Documentation," 2023. [Online]. Available: https://www.nltk.org/
[10] Meta Open Source, "React 18 — New Features and Changes," React Documentation, 2022. [Online]. Available: https://react.dev/blog/2022/03/29/react-v18
[11] Node.js Foundation, "Node.js 20 LTS Release Documentation," Node.js, 2023. [Online]. Available: https://nodejs.org/en/blog/release/v20.0.0
[12] D. Spinellis, "Git," IEEE Software, vol. 29, no. 3, pp. 100–101, 2012.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY