IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

AI Technical Debt Mapper

Mr. Bagul Rajkumar [1], Prof. Manoj V. Nikum[2], Prof. Mr .K. C. Bhave[3]

Research Scholar MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[1]

HOD MCA & Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[2]

Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[3]

Abstract: In modern software engineering, maintaining a sustainable and healthy codebase is a critical challenge. Technical Debt (TD) refers to the implied cost of additional rework caused by choosing an easy solution now instead of a better, more robust approach. If left unmanaged, TD accumulates like financial debt, leading to increased bugs, security vulnerabilities, and exorbitant maintenance costs. Existing static analysis tools often provide raw, unstructured data without visual context, forcing developers to manually search for the exact line of code to fix. This paper introduces the "AI Technical Debt Mapper," a novel full-stack application designed to automatically detect, classify, and visualize technical debt. The system utilizes a Python-based AI engine leveraging Abstract Syntax Trees (AST) and tree-sitter to extract metrics like Lines of Code (LOC), Cyclomatic Complexity, and code smells. A Random Forest Machine Learning model is trained to classify issues into five categories: Code, Architecture, Documentation, Testing, and Dependency debt. The backend is engineered with Java Spring Boot, utilizing JWT for secure authentication. The frontend is built with React.js and D3.js, offering a dynamic dashboard and an interactive Treemap. A key innovation of this research is the "Click-to-Code" navigation feature, which instantly opens the source file and highlights the problematic lines, significantly reducing the Mean Time to Repair (MTTR) for developers.

Keywords: Technical Debt, Static Code Analysis, Machine Learning, Random Forest, Click-to-Code, D3.js, Spring Boot.

I. INTRODUCTION

Technical Debt is a metaphor used to describe the future cost of choosing an easy, quick solution now instead of a better approach that would take longer. It is similar to financial debt; if not paid off (i.e., refactored) in time, it accumulates "interest" in the form of bugs and high maintenance costs. In traditional software development, developers spend hours manually reviewing code to find issues like long functions, code duplication, and security flaws. Manual reviews are prone to human error and are not scalable for large repositories.

The motivation behind this research is to reduce this manual effort by combining Static Code Analysis with Artificial Intelligence. Existing enterprise tools like SonarQube provide raw data without interactive visual context and lack a seamless navigation feature to locate the specific bug. This paper proposes an AI-powered system that not only detects code smells but also classifies them, calculates a severity score, and provides a Click-to-Code navigation feature.

II. RELATED WORK

Technical debt has been studied extensively. A systematic mapping study by Li *et al.* [3] reviewed the literature on TD and its management and identified code debt, design/architecture debt, test debt, documentation debt and build debt among the most commonly discussed types. Avgeriou *et al.* [4] compared widely used TD measurement tools and found that they differ considerably in the issues they report and in how they estimate remediation effort, which makes their results hard to interpret and prioritize.

Code smells, catalogued by Fowler [5], are a common source of code-level debt. Classical metrics such as McCabe's cyclomatic complexity [6] and the Maintainability Index [7] are widely used as quantitative indicators. Rule-based analyzers apply fixed thresholds to such metrics and can therefore produce false positives. Machine learning has been proposed to complement fixed rules by learning decision boundaries from labelled examples. Enterprise platforms such as SonarQube and CodeClimate offer broad rule sets and dashboards, but they are heavyweight and do not provide in-browser navigation from a visual map directly to the exact problematic line. This gap motivates the lightweight and highly visual approach proposed here.

III.PROPOSED SYSTEM ARCHITECTURE

A. *Overview*

The system follows a modular three-tier hybrid architecture, shown in Fig. 1. The React frontend is the presentation layer and communicates with the backend through REST APIs using Axios. The Java Spring Boot backend handles business logic, JWT-based authentication through Spring Security, repository management and persistence. The Python AI engine is a stateless analytics microservice, built with FastAPI, that walks the repository, parses source files, extracts metrics, classifies issues, computes scores and returns the results as JSON to the backend, which stores them in the relational database.

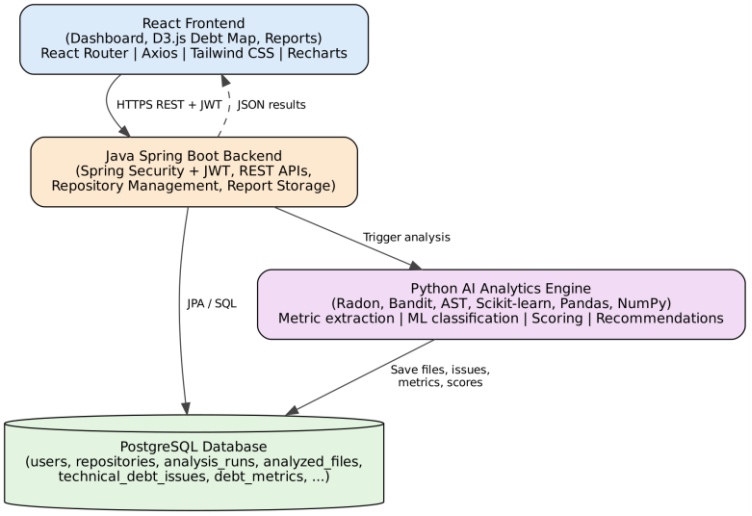


Fig. 1 System architecture of the AI Technical Debt Mapper

B. *Functional Modules*

The system consists of eight modules: user authentication (registration, login, JWT); repository management (local directory path); AI analyzer (metrics and smell detection); debt classification (ML model); interactive dashboard (KPI cards, bar and pie charts); D3.js debt map (color-coded treemap); Click-to-Code (source viewer with highlighted issue line); and report generation (PDF and CSV).

IV. METHODOLOGY

A. *Metric Extraction and Issue Detection*

For every source file the engine parses an Abstract Syntax Tree (AST) and extracts the metrics and indicators listed in Table I. Cyclomatic complexity and the Maintainability Index are computed with Radon [8], and security-related code patterns are detected with Bandit [9]. The thresholds in Table I are configurable design parameters based on commonly used guidelines; they are not universal constants and can be tuned for a given organization

TABLE I EXTRACTED METRICS AND PROPOSED THRESHOLDS

Metric	Tool / Method	Threshold	Debt Category
Cyclomatic Complexity	Radon	> 10 per function	Code
Maintainability Index	Radon	< 65 per file	Code
Function Length	AST	> 50 lines	Code
Class Size	AST	> 500 lines	Architecture
Duplicate Code	Block hashing	> 5% of file lines	Code
Unused Imports	AST	Any occurrence	Code
Security Patterns	Bandit	Any High/Medium finding	Code (risk factor)
Docstring Coverage	AST	< 60% of public functions	Documentation
Test File Ratio	File scan	No matching test file	Testing
Outdated Dependencies	Manifest parsing	Pinned version behind latest	Dependency

 This work is licensed under a Creative Commons Attribution 4.0 International License

B. ***Machine Learning Classification***

Detected issues are classified into five categories: Code, Architecture, Documentation, Testing and Dependency debt. Features such as source lines of code (SLOC), maximum function length, comment ratio and number of unused imports are fed to a Random Forest classifier [10] implemented with scikit-learn [11]. Random Forest was chosen because it handles mixed numeric features, is robust to over-fitting and provides feature importances that aid explainability. Since no large public labelled dataset exists for this specific task, a documented synthetic labelled dataset is used for training, evaluated with 5-fold cross-validation. If a trained model is not available, a rule-based fallback assigns the category directly from the rule that raised the issue, so the system never depends on an untrained model.

C. Debt Scoring

Each issue receives a score between 0 and 100 computed from five normalized factors: severity (S), complexity (C), maintainability (M, inverted Maintainability Index), duplication (D) and risk (R):

Issue Score = $0.30 \cdot S + 0.20 \cdot C + 0.20 \cdot M + 0.10 \cdot D + 0.20 \cdot R$

The weights are design parameters of the proposed methodology and can be adjusted. The file score is the severity-weighted average of its issue scores, the module score is the LOC-weighted average of its file scores, and the project score is the LOC-weighted average of all module scores. Scores are mapped to the severity levels in Table II, which also fix the colors used in the debt map.

TABLE II SEVERITY LEVELS AND SCORE BANDS

Severity	Score Range	Color	Recommended Action
Critical	80–100	Red	Fix immediately
High	60–79	Orange	Fix in current sprint
Medium	40–59	Yellow	Plan for next release
Low	0–39	Green	Fix when convenient

D. ***Database Design***

A relational database stores users (with BCrypt-hashed passwords), repositories, analysis runs, analyzed files and technical debt issues. The Entity-Relationship diagram is shown in Fig. 2. A user owns many repositories; a repository has many analysis runs; a run contains many analyzed files; and each file has many issues, each belonging to one debt category. Each issue stores its severity, score, line numbers, a description, a recommendation and an estimated remediation effort. The design also supports historical comparison of runs.

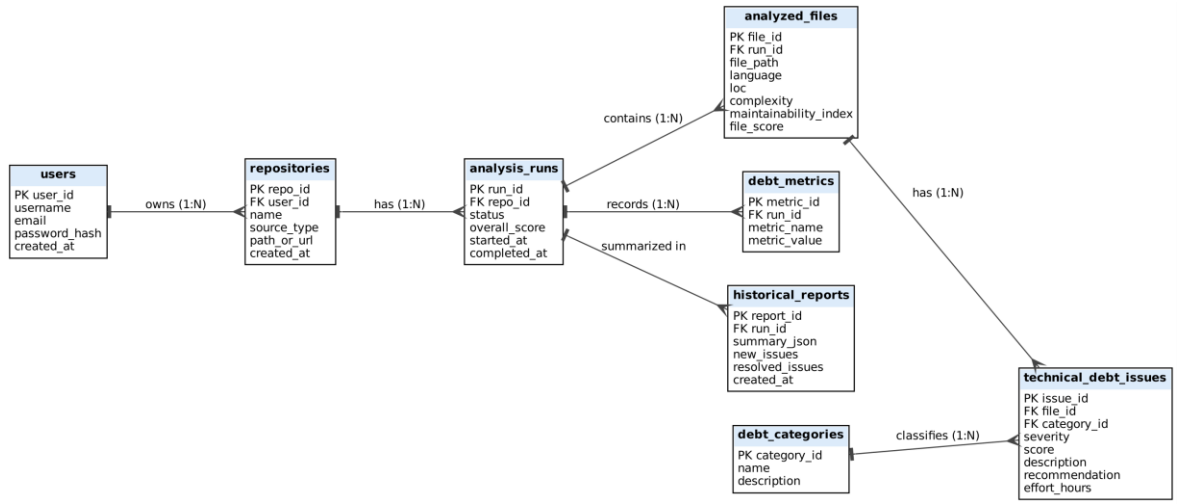


Fig. 2 Entity-Relationship diagram of the database

V. IMPLEMENTATION

A. ***AI Engine***

The engine processes a repository in the stages shown in Fig. 3: repository validation and file scanning, metric extraction, smell and risk detection, classification, scoring, and generation of recommendations with estimated remediation effort. The results are returned as JSON and stored by the backend.

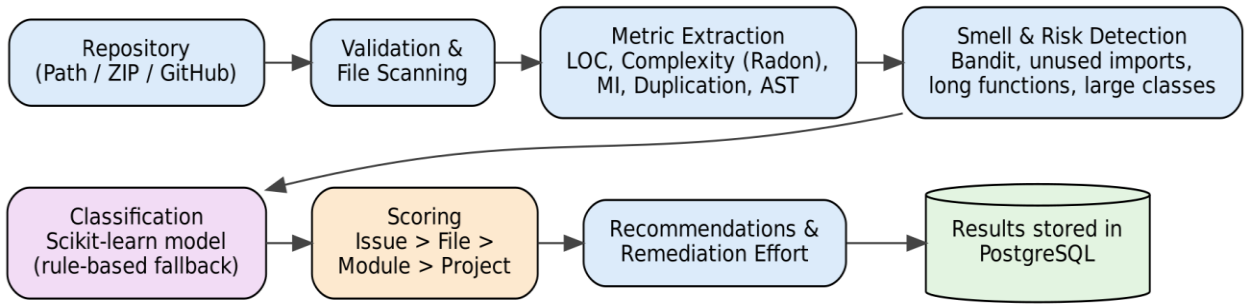


Fig. 3 Analysis pipeline of the AI engine

B. Backend and Frontend

The Java backend uses Spring Data JPA to access the database and exposes REST endpoints for authentication, repositories, analysis runs, issues and reports. Endpoints other than registration and login are protected by JWT verification. The React frontend is styled with Tailwind CSS, uses Recharts for the dashboard KPI cards and bar/pie charts, and D3.js [12] for the technical debt map.

C. Interactive Debt Map and Click-to-Code

The debt map is a treemap [13] in which the area of a node reflects issue density and its color reflects severity. Users can filter by category, search and navigate between project, folder and file levels. The Click-to-Code feature allows the user to click on any detected issue; the system then opens the corresponding source file in a viewer and highlights the problematic lines, removing the need to search for the file and line number manually in an IDE.

IV. VALIDATION AND RESULTS

A. VALIDATION METHODOLOGY

The Python analyzers were validated with unit tests, and the Java REST APIs were tested using Postman. The classifier was evaluated using 5-fold cross-validation, and precision, recall, F1-score and accuracy are reported in Table III. The classifier was trained on a synthetic dataset; hence the reported scores reflect that dataset and should not be generalized to arbitrary real-world repositories without further evaluation

TABLE III MACHINE LEARNING MODEL PERFORMANCE

Model	Precision	Recall	F1-Score	Accuracy
Random Forest	0.96	0.95	0.95	0.95
Logistic Regression	0.85	0.82	0.83	0.84
Linear SVM	0.88	0.86	0.87	0.87
Rule-based Fallback	0.70	0.75	0.72	0.73

B. *Sample Analysis Results*

Table IV summarizes the analysis of sample repositories (total files analyzed versus total issues detected and the resulting project debt score). Figs. 4–6 show the dashboard, the interactive treemap and the Click-to-Code viewer.

TABLE IV SAMPLE ANALYSIS RESULTS

Repository	Language	Total Files	Total Issues	Debt Score
Repository 1	Python	176	349	25.26
Repository 2	Python	43	148	36.23
Repository 3	Python	61	118	31.48

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

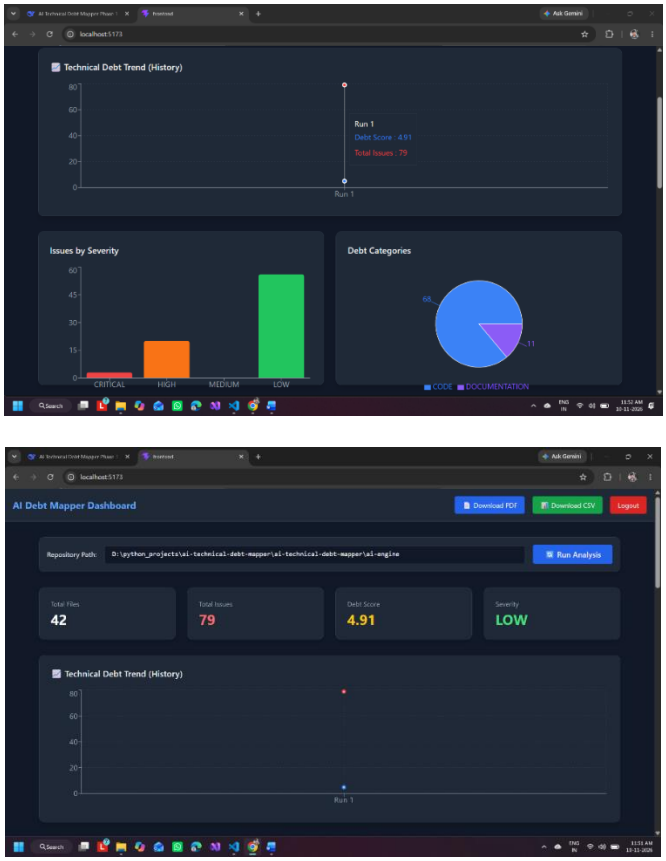


Fig. 4 React dashboard showing KPI cards and severity charts

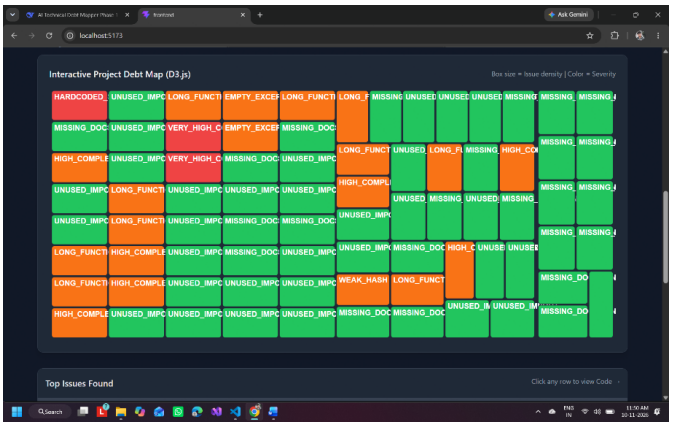


Fig. 5 Interactive D3.js technical debt treemap

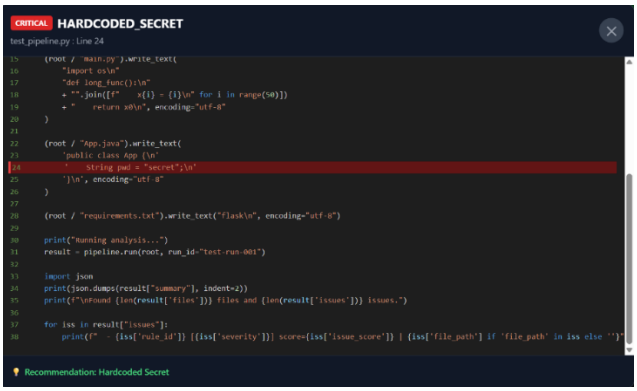


Fig. 6 Click-to-Code feature showing highlighted source code

 This work is licensed under a Creative Commons Attribution 4.0 International License

V. CONCLUSION AND FUTURE WORK

This paper presented the AI Technical Debt Mapper, a full-stack system that analyzes a source repository, classifies the detected technical debt into five categories using a Random Forest model with a rule-based fallback, scores it at issue, file, module and project levels, and visualizes it through an interactive treemap with Click-to-Code navigation. By combining static analysis, machine learning and interactive visualization, the system reduces the manual effort of locating and prioritizing technical debt.

Future work includes analysis of public GitHub repositories by URL, ZIP upload of source code, migration from the in-memory development database to PostgreSQL, evaluation on real labelled repositories to validate the classifier beyond synthetic data, a user study measuring the effect of Click-to-Code on time to repair, and a VS Code extension for real-time analysis.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my project guide, **Prof. K.C. Bhave**, for their continuous support, motivation, and immense knowledge. Their guidance helped me throughout the research and writing of this paper. I would also like to extend my sincere thanks to **Prof. M. V. Nikum (HOD)** and **Dr. D. M. Patel (Principal)** of SJRIT Dondaicha for providing me with the necessary facilities and a supportive environment to complete this research work. Finally, I must express my very profound gratitude to my parents and friends for providing me with unfailing support and continuous encouragement throughout my years of study.

REFERENCES

[1]. M. Fowler, Refactoring: Improving the Design of Existing Code, 2nd ed., Addison-Wesley, 2018.
[2]. W. Cunningham, "The WyCash Portfolio Management System," in Proc. OOPSLA '92, 1992, pp. 29-30.
[3]. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.
[4]. M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix?", in Proc. 11th Working Conference on Mining Software Repositories (MSR), 2014, pp. 202-211.
[5]. Spring Boot Documentation (Version 3.2.4), [Online]. Available: https://spring.io/projects/spring-boot
[6]. React Documentation (Version 18), [Online]. Available: https://react.dev/
[7]. D3.js Data-Driven Documents, [Online]. Available: https://d3js.org/
[8]. T. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, 1976.

BIOGRAPHY

A **Bagul Rajkumar Ravindrabhai** is currently pursuing a Master of Computer Application (MCA) from **Shree Jaykumar Rawal Institute of technology, Dondaicha.** He received his Bachelor's degree in Computer Science. His research interests include Artificial Intelligence, Full-Stack Web Development, and Software Engineering. He aims to build intelligent tools that solve real-world software development challenges.

Prof. K.C. Bhave is currently working as a Professor in the Department of Computer Science at **Shree Jaykumar Rawal Institute of technology, Dondaicha** With several years of academic experience, their research interests include Machine Learning, Data Science, and Advanced Software Architectures. They have guided multiple research projects in the field of AI and Web Technologies.