BugScope: Intelligent Prediction of High-Risk Files in Software Projects

Bazgour Yassine[1], Abouhane Zahra[1], and Mjahd Soukaina[1]

[1]Semlalia Faculty, Department of Computer Science, Cadi Ayyad University, Marrakesh, Morocco

BugScope is an AI-powered system designed to predict high-risk files in software projects by analyzing code metrics such as complexity, size, and structure. Leveraging the NASA PROMISE CM1 dataset, we trained a Random Forest classifier to automatically classify files as safe, moderate risk, or risky. The system integrates seamlessly with a Flask-based web interface, allowing users to paste code, upload files, or provide GitHub repository links for automated analysis. BugScope facilitates proactive software quality assurance by prioritizing testing and code review efforts based on predicted defect risk, enhancing development efficiency and reliability. The project repositories are publicly available at [1] and [2].

1. INTRODUCTION

Software bugs cause significant costs and slow down development. Automatically predicting high-risk files allows testing and maintenance efforts to focus on the most vulnerable modules. This project aims to build a system capable of identifying files likely to contain defects based on code metrics, using machine learning techniques to improve the reliability and efficiency of the software development process.

2. PROJECT CONTEXT

In modern software development, identifying defective code early is critical to reducing maintenance costs and improving software quality. Traditional testing approaches can be time-consuming and may not cover all potential problem areas. This project focuses on predicting high-risk files in a software project by analyzing code metrics, such as complexity, size, and structure. Using the NASA CM1 dataset [3], which contains labeled examples of defective and clean files along with various static code metrics, we aim to train a machine learning model to automatically flag files likely to contain bugs. The approach integrates well with agile development practices, such as SCRUM, by allowing teams to prioritize testing and code review efforts based on predicted risk, ultimately improving the efficiency of the development lifecycle.

3. ANALYSIS AND DESIGN

A. Analysis

A.1. Problem understanding

Modern software projects often contain large codebases with many modules. Some of these modules are more prone to defects than others, but identifying them manually is time-consuming and error-prone. The objective of this system is to analyze software metrics extracted from source code files and predict which modules are likely to be defective. By prioritizing risky components early, development teams can optimize testing effort, reduce maintenance cost, and improve software reliability. This project uses the PROMISE CM1 dataset, which includes NASA spacecraft flight software metrics labeled as defective or non-defective, allowing supervised learning to build a predictive model

A.2. Stakeholders

- Software Engineers: Receive defect risk reports to guide debugging and refactoring.
- Quality Assurance Team: Prioritize testing efforts toward high-risk modules.
- Project Managers: Improve cost planning and defect resolution strategies.
- Researchers / Data Scientists: Benchmark and extend defect prediction models.
- End Users: Benefit from improved software reliability.

A.3. Functional requirements

1. Import dataset.
2. Train a supervised classification model to predict defective modules.
3. Evaluate performance using metrics such as accuracy, precision, recall, F1-score, Macro Avg, Weoghted Avg and Confuson Matrix.
4. Predict risk labels on unseen source code metric files.
5. Display prediction results and evaluation scores.

A.4. Non-functional requirements

1. Performance: The model must provide predictions within acceptable time for medium-sized projects (hundreds of modules).
2. Accuracy: The system should achieve an evaluation metric competitive with reported literature on the PROMISE dataset.
3. Usability: Predictions and reports must be easy to understand by engineers.
4. Maintainability: Code must be modular to allow updating the model or dataset.
5. Reproducibility: Same data and configuration should yield identical results.

A.5. Product backlog (user stories)

- As a user, I want to access a web interface so I can interact with the defect prediction system.
- As a user, I want to upload a source code file so the system can analyze it.
- As a user, I want to paste raw code text so the system can evaluate it without file upload.
- As a user, I want to provide a GitHub repository link so the system can fetch and analyze its files.
- As a user, I want to view the predicted defect risk for the submitted code so I know whether it is risky.
- As a user, I want to download or view a summary report so I can share results or take action.
- As a user, I want visual indicators (e.g., color, score) so I can easily understand risk severity.
- As a user, I want system feedback if my upload or pasted input is invalid so I can correct it.

A.6. Sprint backlog planning

Sprint 1: UI Access
Focus: provide the interface the user can reach and interact with.

- Design a minimal web interface structure (HTML + Flask routing).
- Implement homepage page with accessible navigation.
- Display instructions for users.

Sprint 2: Code Input Functionality
Focus: allow users to submit software code for analysis.

- Implement file upload handling.
- Implement a text area to paste raw source code.
- Implement GitHub link submission input.
- Validate file type and input correctness.
- Display error messages when input is invalid.

Sprint 3: Prediction and Result Display
Focus: provide model-based analysis and feedback.

- Extract code metrics from submitted files or pasted code.
- Connect Flask backend to the trained ML model.
- Run prediction to classify files as risky or not.
- Display prediction results clearly to the user.
- Add visual risk indicator (colors or score).

Sprint 4: Reporting and Enhancement
Focus: allow users to obtain interpretable reports.

- Generate a summary report of prediction results.
- Render report in the web interface.
- Allow users to download or view the report.
- Perform UI polishing and layout improvements.

B. Design

B.1. System architecture

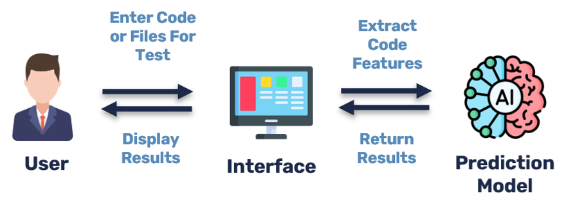

Fig. 1. User sinario using BugScope interface

B.2. UML diagrams

- Use case diagram

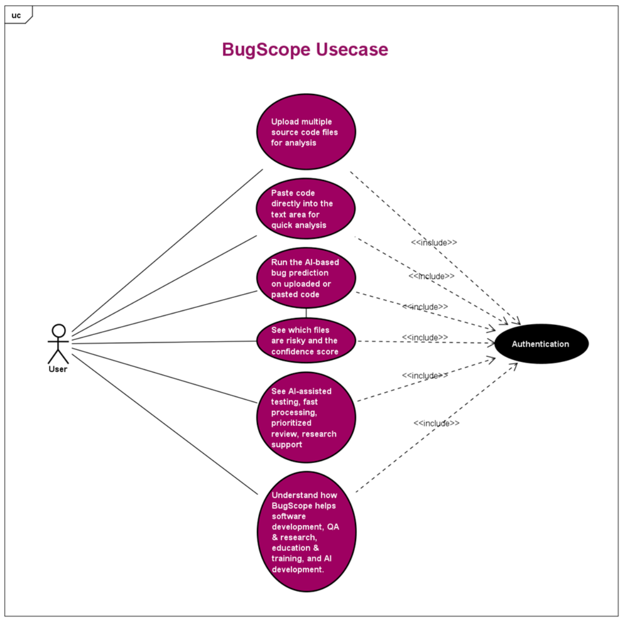
uc
BugScope Usecase
Upload multiple source code files for analysis
Paste code directly into the text area for quick analysis
Run the AI-based bug prediction on uploaded or pasted code
See which files are risky and the confidence score
See AI-assisted testing, fast processing, prioritized review, research support
Understand how BugScope helps software development, QA & research, education & training, and AI development.
User
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
Authentication

Fig. 2. Usecase Diagram

- Deployment or architecture diagram

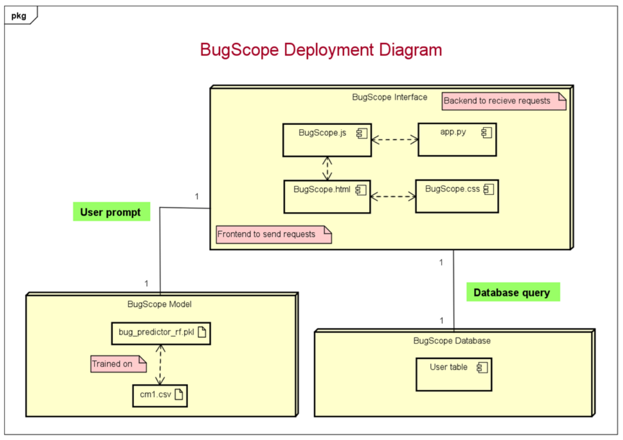

Fig. 3. Deployement Diagram

4. PROPOSED APPROACH

In this project, the proposed approach integrates machine learning techniques with a web interface to automatically predict risky files in a software project. The pipeline is designed to allow the user to provide code through three options: paste code directly, upload a file, or provide a GitHub repository path. The system then evaluates the code using a pre-trained Random Forest model and returns a prediction indicating the likelihood of the file being defective.

A. Data Acquisition and Preprocessing

The model is trained on the **NASA PROMISE CM1 dataset**, which contains labeled software files along with static code metrics such as Lines of Code (LOC), cyclomatic complexity, and other relevant measurements. The preprocessing pipeline includes:

- Handling missing or inconsistent metric values.
- Normalizing numerical features to ensure fair contribution to the model.
- Splitting the dataset into training and validation sets.

B. Feature Selection

The system leverages key software metrics that correlate with defect proneness:

- Lines of Code (LOC)
- Cyclomatic Complexity (v(g))
- Essential Complexity (ev(g))
- Module Design Complexity (iv(g))
- Coupling Metrics (n, v, l, d, i)

C. Model Training

The Random Forest classifier is selected for its robustness and interpretability. Key aspects include:

- Training on labeled CM1 dataset to classify files as defect-prone or clean.
- Tuning hyperparameters (number of trees, max depth) to optimize accuracy.
- Evaluating model performance using metrics such as accuracy, F1-score, Precision and Recall.

D. Integration with Web Interface

The trained model is integrated into a simple Flask-based interface. Users can interact with the system via:

- **Paste code**: directly input code in a text box.
- **Upload file**: provide a source code file for analysis.
- **GitHub repository**: specify a repository URL for automatic extraction of code files.

After processing the code, the interface calls the Random Forest model to predict the risk score for each file and displays results in an easily interpretable format. These features are extracted from the file to predict defects automatically.

E. Model Training

F. Evaluation

The approach includes continuous evaluation:

- Testing model predictions on unseen files.
- Displaying metrics (accuracy, precision, recall) to validate model reliability.
- Allowing iterative improvement of both the model and the interface based on user feedback.

5. IMPLEMENTATION

In this project, we developed BugScope, an AI-powered system to predict potentially risky files in a software project. By leveraging the CM1 dataset and training a Random Forest model, we were able to automatically extract code metrics and classify files as safe, moderate risk, or risky with high confidence. The project demonstrates the practical application of machine learning for software quality assurance, providing developers with an efficient tool to focus on critical parts of the code and reduce potential defects. The implemented interface allows easy file uploads and instant predictions, making the system accessible and user-friendly. Overall, BugScope showcases how AI can enhance software reliability and support proactive code review processes.

6. CONCLUSION

A. Summary

This project demonstrates a practical approach to automatically predicting defect-prone files in software projects. By leveraging static code metrics from the NASA PROMISE CM1 dataset and a Random Forest classifier, we provide an interface that allows users to check files for potential bugs through code paste, file upload, or GitHub repository input. The system helps prioritize code review and testing efforts, integrating seamlessly with agile development practices like SCRUM.

B. Limitations

While the model shows promising results, several limitations exist:

- The current model is trained on a single dataset (CM1), which may limit generalization to other projects or programming languages.

- Predictions rely solely on static code metrics and do not consider runtime behavior or dynamic testing results.
- The interface, while functional, is minimal and could benefit from more detailed visualization of results and risk factors.

C. Future Improvements

Potential improvements include:

- Extending the training dataset to include multiple PROMISE datasets or other open-source projects to improve model generalization.
- Experimenting with more advanced machine learning models, such as gradient boosting or deep learning approaches, to enhance prediction accuracy.
- Enhancing the web interface with richer feedback, detailed risk explanations, and batch file analysis capabilities.
- Incorporating dynamic analysis metrics to complement static metrics for more comprehensive risk assessment.

REFERENCES

1. M. S. Bazgour Yassine, Abouhane Zahra, "Bugscope server: Backend for high-risk file prediction in software projects," https://github.com/DarttGoblin/BugScope_server (2025). Accessed: 2025-12-07.
2. M. S. Bazgour Yassine, Abouhane Zahra, "Bugscope: Web interface for high-risk file prediction in software projects," https://github.com/DarttGoblin/BugScope (2025). Accessed: 2025-12-07.
3. R. U. Haque, "Cm1 software defect dataset," https://www.kaggle.com/datasets/radowanulhaque/software-defect?select=cm1.csv (2025). Accessed: 2025-12-07.