Automated Programming Evaluation using MERN

Kshama Patel
Dept. of Information Technology
DAIICT
Gandhinagar, Gujarat
202312002@daiict.ac.in

Jetal Savani
Dept. of Information Technology
DAIICT
Gandhinagar, Gujarat
202312117@daiict.ac.in

Rupesh Singh
Dept. of Information Technology
DAIICT
Gandhinagar, Gujarat
202312124@daiict.ac.in

***Abstract*—Automated programming evaluation has become crucial in educational institutions to provide timely feedback and maintain grading consistency. This paper presents a MERN (MongoDB, Express.js, React.js, and Node.js) stack-based system that automates programming evaluation, supporting multiple languages and providing real-time feedback. By integrating front-end, back-end, and database functionalities, this system allows for scalable, role-based access for students, faculty, and administrators, filling gaps in traditional educational management systems and simplifying grading processes for programming courses.**

***Index Terms*—Automated grading, MERN stack, educational technology, programming evaluation, real-time feedback**

I. Introduction

Educational institutions are increasingly adopting automated grading systems to address the limitations of manual evaluation, such as time-intensity and subjectivity. Current educational management systems like Moodle lack specialized tools for programming assessment, often requiring manual grading or external integration with other platforms. The MERN stack-based solution presented here aims to streamline programming evaluation through real-time, automated grading that supports multiple languages and provides immediate feedback to students.

II. Static Code Analysis for C++ Employee Management System

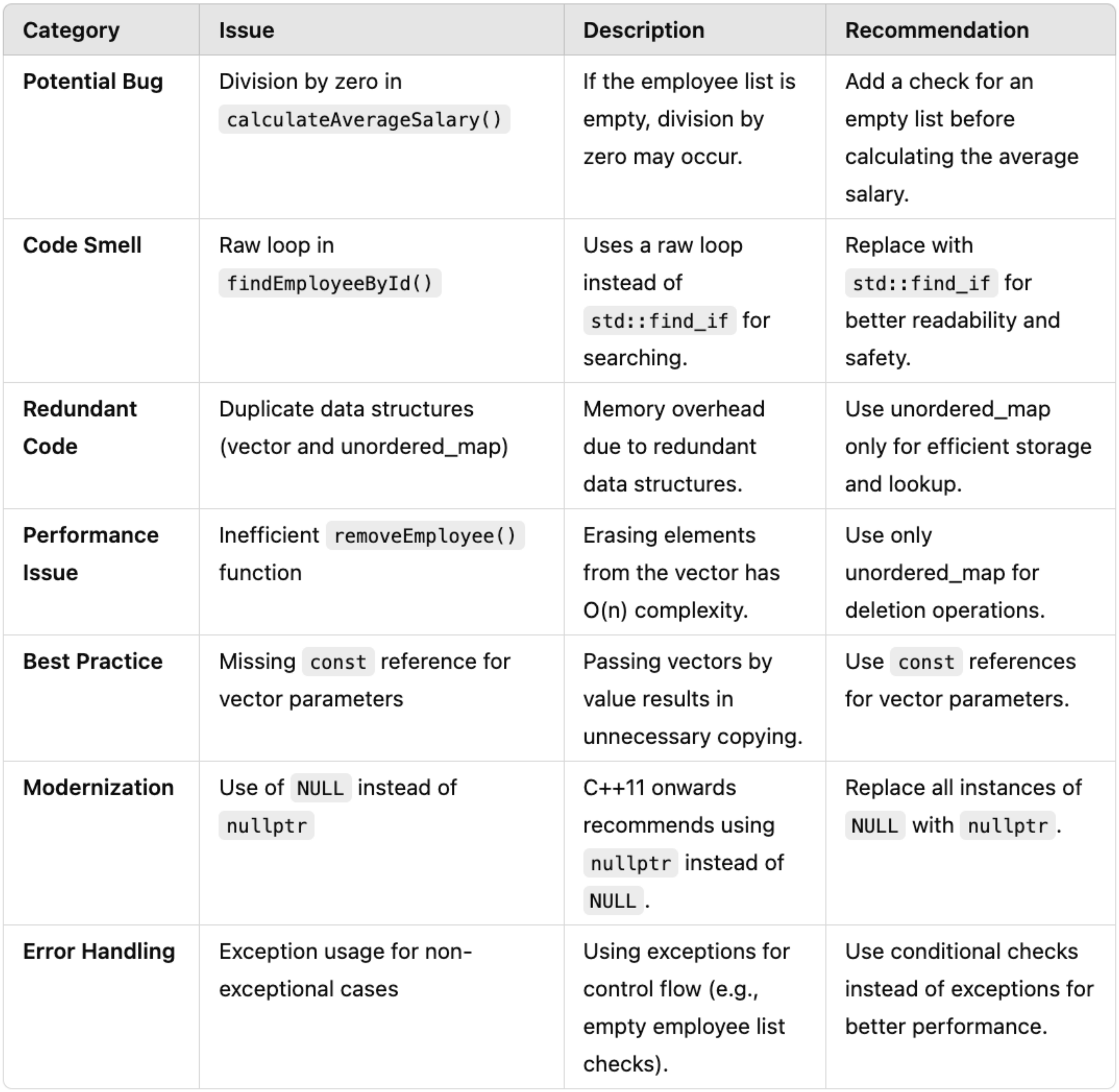
A. Title: Static Code Analysis for a C++ Employee Management System

Abstract: In this research, we apply static code analysis to a C++ program implementing an employee management system. The analysis was conducted using tools like Cppcheck and Clang-Tidy to detect potential bugs, code smells, redundant code, performance issues, and best practice violations. This study highlights common problems that can impact code quality, maintainability, and performance, and offers recommendations for improvement.

B. Analysis Process

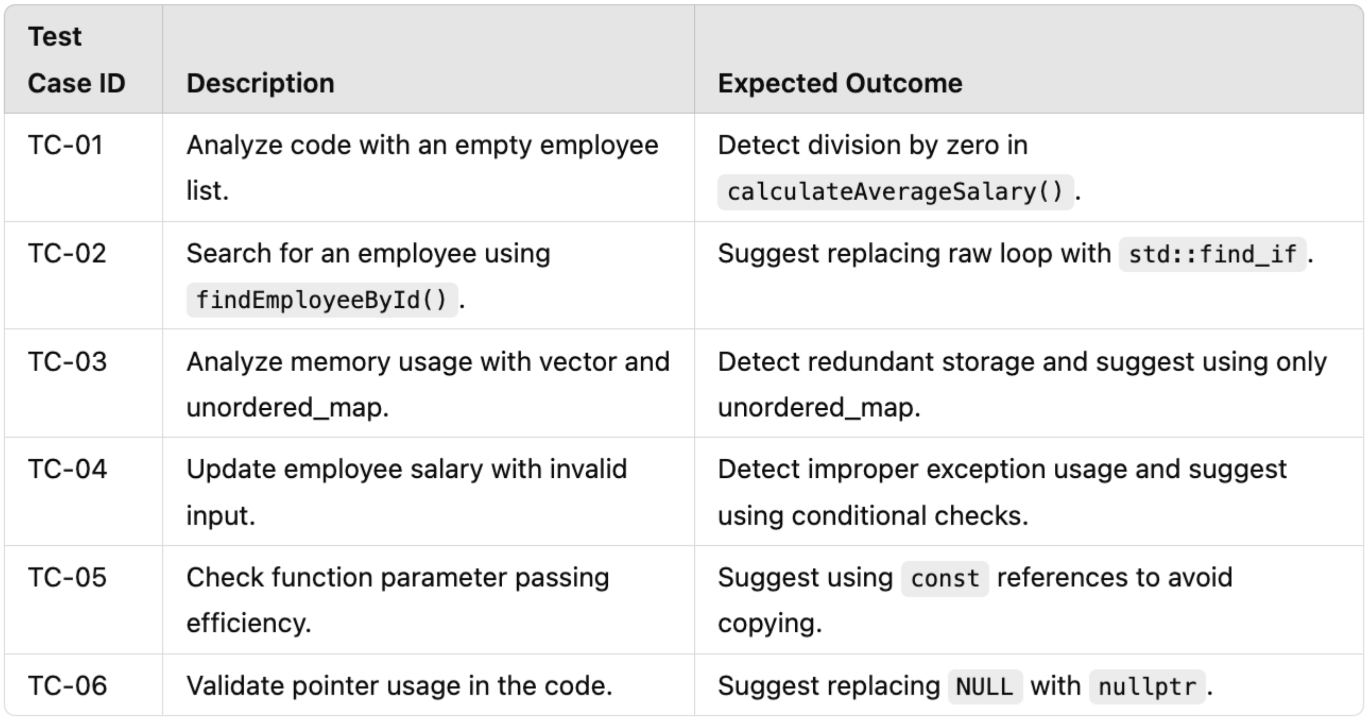
The C++ code was thoroughly analyzed using standard static analysis tools, covering a range of issues. The following types of problems were identified:

- **Potential Bug:**
  - Division by Zero: In the function `calculateAverageSalary()`, division by zero may occur if the employee list is empty.
  - Improper Exception Handling: Exceptions are used for regular control flow, such as checking if the list is empty.
- **Code Smell:**
  - Raw Loop Usage: The function `findEmployeeById()` uses a raw loop instead of utilizing `std::find_if` from the standard library.
- **Redundant Code:**
  - Duplicate Data Structures: The employee data is stored in both a vector and an unordered_map, leading to increased memory usage and redundancy.
- **Performance Issue:**
  - Inefficient Function Calls: The `removeEmployee()` function uses `std::remove_if` with a vector, resulting in O(n) complexity.
- **Best Practice:**
  - Missing Const References: Functions that take large containers like vectors as parameters do not use const references, causing unnecessary copying of data.
- **Modernization:**
  - Use of NULL Instead of nullptr: The code uses NULL for pointer initialization, which is outdated. Modern C++ standards recommend using `nullptr`.
- **Error Handling:**
  - Exception Usage for Non-Exceptional Cases: The code throws exceptions for cases that can be handled with simple conditional checks.[7]



Category	Issue	Description	Recommendation
Potential Bug	Division by zero in `calculateAverageSalary()`	If the employee list is empty, division by zero may occur.	Add a check for an empty list before calculating the average salary.
Code Smell	Raw loop in `findEmployeeById()`	Uses a raw loop instead of `std::find_if` for searching.	Replace with `std::find_if` for better readability and safety.
Redundant Code	Duplicate data structures (vector and unordered_map)	Memory overhead due to redundant data structures.	Use unordered_map only for efficient storage and lookup.
Performance Issue	Inefficient `removeEmployee()` function	Erasing elements from the vector has O(n) complexity.	Use only unordered_map for deletion operations.
Best Practice	Missing `const` reference for vector parameters	Passing vectors by value results in unnecessary copying.	Use `const` references for vector parameters.
Modernization	Use of `NULL` instead of `nullptr`	C++11 onwards recommends using `nullptr` instead of `NULL`.	Replace all instances of `NULL` with `nullptr`.
Error Handling	Exception usage for non-exceptional cases	Using exceptions for control flow (e.g., empty employee list checks).	Use conditional checks instead of exceptions for better performance.

Fig. 1. Summary of Static Code Analysis Issues[7]



Test Case ID	Description	Expected Outcome
TC-01	Analyze code with an empty employee list.	Detect division by zero in `calculateAverageSalary()`.
TC-02	Search for an employee using `findEmployeeById()`.	Suggest replacing raw loop with `std::find_if`.
TC-03	Analyze memory usage with vector and unordered_map.	Detect redundant storage and suggest using only unordered_map.
TC-04	Update employee salary with invalid input.	Detect improper exception usage and suggest using conditional checks.
TC-05	Check function parameter passing efficiency.	Suggest using `const` references to avoid copying.
TC-06	Validate pointer usage in the code.	Suggest replacing `NULL` with `nullptr`.

Fig. 2. Test Cases for Verification [7]

III. Results

The static analysis identified seven key issues across the categories of potential bugs, code smells, redundant code, performance concerns, best practices, modernization, and error handling. Addressing these problems with the provided recommendations can significantly improve the code's quality, performance, and compliance with modern C++ standards.

IV. System Architecture and Technology Stack

The MERN-based system is built on a three-tier architecture, integrating MongoDB for data storage, Express.js and Node.js for server-side processing, and React.js for a dynamic user interface.

A. MongoDB

MongoDB is used for document-based data storage, supporting efficient retrieval and handling of student records, grades, and assignment data in a flexible, schema-less format.

B. Express.js and Node.js

Express.js and Node.js form the back-end server, processing HTTP requests, managing data interactions, and handling test-case-based evaluation for programming assignments.

C. React.js

React.js powers the client-side interface, providing an interactive platform for students, faculty, and admins. It supports a virtual DOM for efficient updates, critical for real-time feedback.

V. System Modules and Features

A. Admin Module

The admin module manages user roles, including adding faculty and student profiles, monitoring system performance, and overseeing data integrity.

B. Faculty Module

Faculty can create assignments, view student performance, and manage evaluation metrics. The module supports test-case based grading to ensure code correctness and logical accuracy.

C. Student Module

Students can submit assignments and receive immediate feedback. The system provides a breakdown of test-case performance, allowing students to identify areas for improvement.

D. Compiler Module

The compiler module supports multiple languages and provides real-time test-case evaluation. This integration allows for automated, accurate grading based on defined test cases.

VI. Methodology

The grading process involves test-case based evaluation, where code submissions are automatically assessed for correctness and partial correctness. The system's asynchronous processing supports high volumes of submissions, ensuring efficient grading even with large class sizes.

VII. Results and Performance Analysis

Preliminary tests show that the MERN-based system provides quick, consistent feedback, reducing grading time compared to manual methods. The system supports simultaneous submissions with minimal latency, demonstrating scalability in high-enrollment courses.

VIII. Discussion

The MERN-based automated programming evaluation tool presents a scalable solution that improves the grading experience for both students and educators. By combining real-time feedback, multi-language support, and role-based access, it addresses gaps in existing educational management systems.

IX. Conclusion

This paper presents a MERN stack-based automated programming evaluation system tailored for educational use. Future work includes integrating AI-based feedback and expanding language support, enhancing the system's applicability across various programming courses.

Acknowledgment

The authors would like to thank DAIICT, Gandhinagar, for providing the infrastructure and resources required for the development of this project. Special thanks to the faculty advisors for their valuable insights and guidance.

References

[1] H. Aldriye, et al., "Automated Grading Systems for Programming Assignments," IJACSA, vol. 10, no. 3, 2019.

[2] S. I. Malik, et al., "PROBSOL: A Web-Based Application for Introductory Programming," Springer Nature, 2019.

[3] S. Patil, et al., "College ERP Using MERN Stack," Int. J. of Scientific Research in Computer Science, 2021.

[4] V. Aggarwal and J. Verma, "Comparative Analysis of MEAN and MERN Stacks," Int. J. of Recent Research Aspects, vol. 5, 2018.

[5] S. Papandrea, et al., "A Support Platform for Computer Programming Education," Springer International, 2018.

[6] T. Szopinski and K. Bachnik, "Student Evaluation of Online Learning During COVID-19," Technological Forecasting and Social Change, vol. 174, 2022.

[7] Github Code Link: https://github.com/JetalSavani/Automated-Programming-Evaluation-using-MERN