Educational Data Mining for Assessing Student Code Quality in Programming Courses

Devanshi Prajapati, Software Engineering
Mentor: Ruben Acuña, Assistant Teaching Professor
School of Computing and Augmented Intelligence

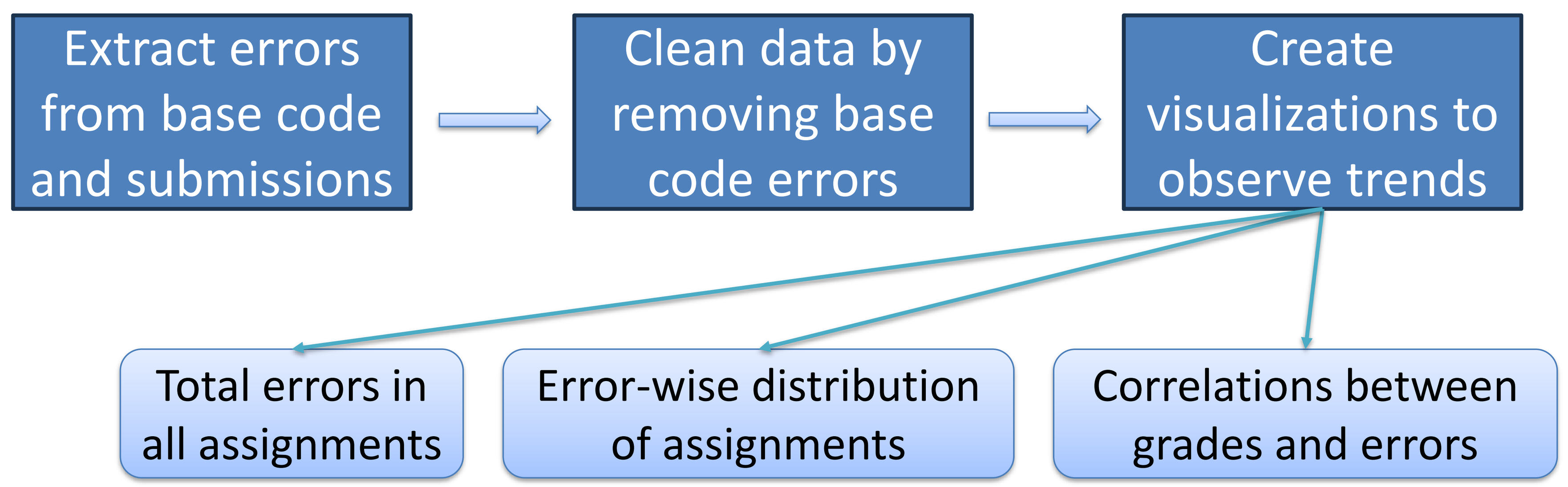
Research Question

To develop a data-driven framework that analyzes student code submissions, and extracts insights, enabling instructors to effectively teach industry-standard coding practices and better prepare students for professional software engineering careers.

Introduction

This research project aims to enhance automated assessment in programming courses by integrating educational data mining techniques with autograders in a Data Structures & Algorithms course. The study analyzes performance metrics and code quality indicators from student Java assignments in SER222 using static analysis tools. By examining trends in error categories, correlating measures with grades, and tracking code quality evolution over time, the project seeks to provide instructors with valuable insights to improve assessment and teaching of code quality. The work so far has focused on collecting assignment data, cleaning it, and creating visualizations to identify basic trends between error categories and programming assignments. This research will enable more effective evaluation of non-functional requirements in student code.

Workflow



Understanding Good vs Bad Code

```java
/* Utility class for basic math operations. */
public class MathUtils {
    // Method to calculate the sum of even numbers in an array
    public int sumEvenNumbers(int[] numbers) {
        return Arrays.stream(numbers).filter(num -> num % 2 == 0).sum();
    }
    /** Method to calculate the factorial of a number */
    public int factorial(int number) {
        if (number < 0) {
            throw new IllegalArgumentException(s:"Number must be non-negative.");
        }
        return IntStream.rangeClosed(1, number)
                        .reduce(1, (a, b) -> a * b);
    }
}
```

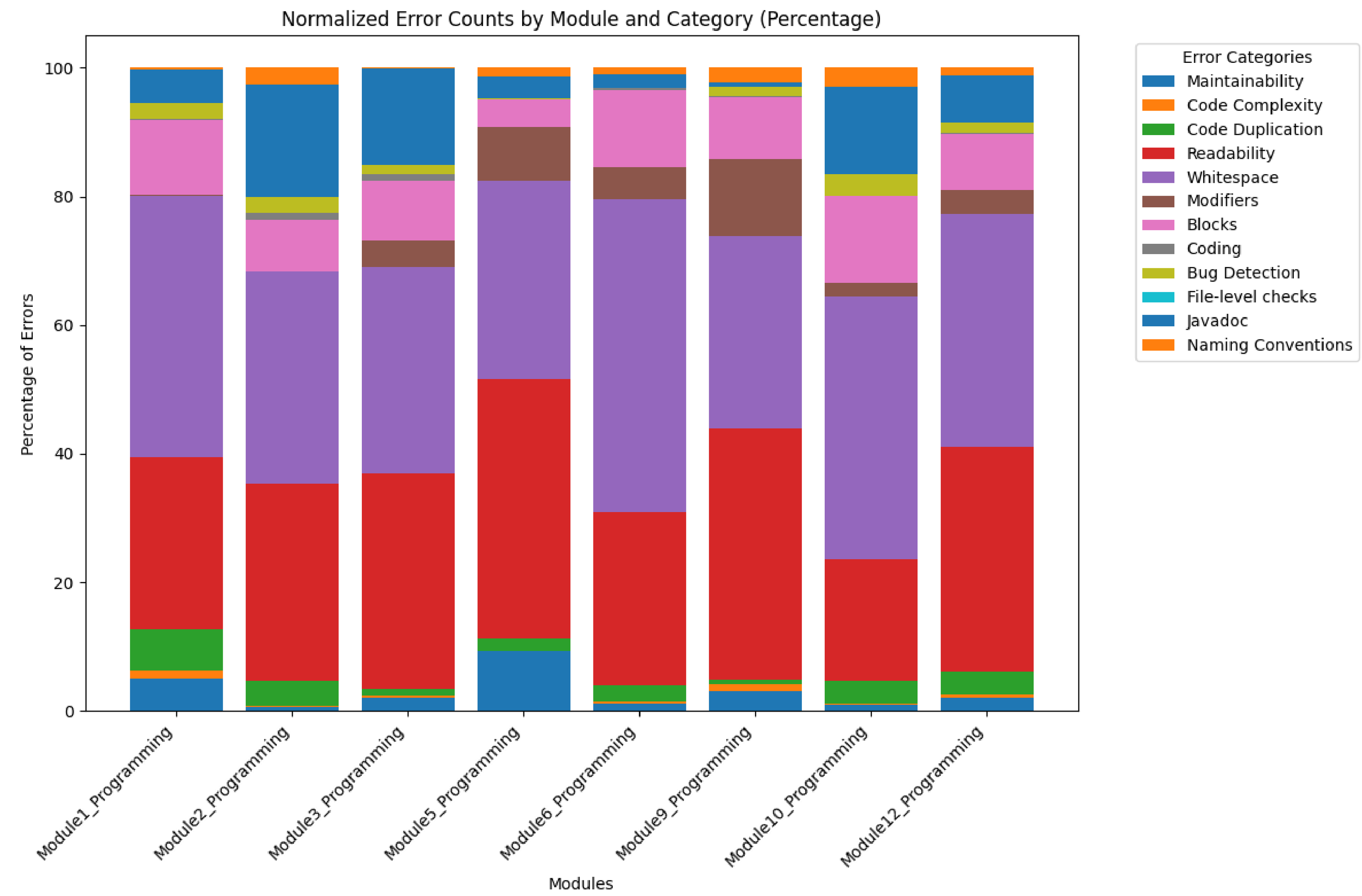
Fig. 1 Good Code

Category	Good Code (Fig. 1)	Bad Code (Fig. 2)
Maintainability	Modular design, easy to extend	Less modular, harder to maintain
Code Complexity	Uses streams for concise logic	Manual loops increase complexity
Code Duplication	No duplication observed	Potential for duplication in loop logic
Readability	Clear method names, concise code	Unclear method names (sum, fact)
Whitespace	Proper indentation and spacing	Inconsistent spacing
Modifiers	Public access modifier used	No access modifiers specified
Blocks	Consistent block structure	Adequate block structure
Coding	Uses modern Java features (streams)	Uses older, more verbose coding style
Bug Detection	Error handling in factorial method	No input validation or error handling
Javadoc	Well-documented with Javadoc	Lacks documentation and comments
Naming Convention	Follows standard Java conventions	Poor naming (sum, fact)

```java
public class BadCodeExample {
    public int sum(int[] arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] % 2 == 0) {
                sum += arr[i];
            }
        }
        return sum;
    }
    public int fact(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

Fig. 2 Bad Code

Preliminary Results



Inferences

The raw error counts are converted into relative percentages for each module. This allows comparison across modules by making each bar represent 100% of the total errors, with segments showing the proportion of each error category.

- **Naming Conventions (Red) and Readability (Purple)** errors are consistently high, indicating students struggle with these aspects across all modules.
- **Whitespace (Pink) and Code Complexity (Brown)** errors are notable, suggesting issues with formatting and overly complex code.
- **Javadoc (Orange) and Modifiers (Blue)** errors are minimal, showing students generally follow documentation and access control practices.
- **Bug Detection (Yellow) and Maintainability (Dark Blue)** errors are low, indicating fewer bugs but room for improvement in maintainability.

This analysis highlights areas where students consistently struggle, such as Naming Conventions and Readability, allowing instructors to focus on improving these aspects through targeted feedback and additional resources. By addressing common issues like Whitespace and Code Complexity, instructors can refine their course content to better align with industry standards, ultimately enhancing student learning outcomes.

MORE
Masters Opportunity for Research in Engineering

ASU Ira A. Fulton Schools of Engineering
Arizona State University