Mini-Project using Java

Logic-Flow Code Visualiser

Project Synopsis

Sr. No.	Name	PRN	Roll No.
1	Prakul Singh	1032233410	44
2	Vedika Ramdas Rana	1032233559	49

Submitted to- Prof. Suhas Joshi

MIT World Peace University

1. Introduction

In modern computer science curricula, evaluating student programming assignments often falls into the trap of "Black-Box" testing where a submission is judged solely on whether it passes test cases. Our project, Logic-Flow Code Visualizer (LFCV), is a comprehensive Java-based framework designed to provide our professors with a "White-Box" perspective. By transforming raw source code into an Abstract Syntax Tree (AST) and a Control Flow Graph (CFG), our system allows evaluators to see the structural intent and logic quality behind the output.

Our goal is to bridge the gap between functional correctness and qualitative design, ensuring that students are writing efficient, maintainable, and original code.

2. Problem Statement

Professors currently face significant hurdles when grading large batches of programming assignments.

- **Manual Inspection Fatigue:** Evaluating the logic flow of hundreds of files manually is prone to error and exhaustion.
- **Invisible "Spaghetti" Code:** Students can pass functional tests while using deeply nested, inefficient, or "hardcoded" logic that violates clean coding principles.
- **Plagiarism Evolution:** Traditional text-based detection is easily bypassed by renaming variables, but this project focuses on the logic structure which is harder to disguise.
- **Lack of Objective Quality Metrics:** There is a need for a centralized tool that provides quantitative data on code complexity to justify grading decisions.

3. Objectives

The main objectives of our project are:

A. To design a centralized evaluator's suite that automates the qualitative analysis of Java source code.

B. To visualize execution paths through node-based graphs, allowing our professors to spot logical flaws at a glance.

C. To calculate objective metrics such as Cyclomatic Complexity and Nesting Depth automatically.

D. To implement "Logic Fingerprinting" to ensure academic integrity by comparing structural patterns across submissions.

E. To provide a streamlined reporting tool that transforms complex code analysis into a simple "Grade-Support" summary.

4. Scope of project

Our system is specifically built to handle the unique requirements of an academic environment:

- **Target Audience:** Exclusively for Professors, Lab Assistants, and Evaluators.
- **Language Support:** Optimized for Java 17+ (CS101 through Advanced Data Structures levels).
- **Environment:** Designed as a lightweight desktop application with a focus on batch-processing student directories.
- **Scalability:** System can handle everything from a single code snippet to an entire semester's worth of student submissions in a single run.

5. Proposed System Architecture

A. Presentation Layer: A minimalist, data-heavy JavaFX interface designed for rapid review and graph interaction.

B. Analysis Layer: The core "Engine" that utilizes JavaParser for AST generation and JGraphT for mathematical flow modelling.

C. Data Access Layer: A local repository (JSON or SQLite) to store "Logic Hashes" and historic complexity benchmarks for the entire class.

6. Functional Modules in Detail

Project is broken down into specific modules that serve as the main aide for evaluators:

I. Static Analysis & AST Module
   - A. Structural Parsing: Converts student code into a tree of logical nodes.
   - B. Metadata Extraction: Identifies method signatures, loop types, and decision points without executing the code.

II. Control Flow Visualization Module
   - A. Graph Rendering: Automatically generates a flowchart of every method.
   - B. Complexity Heat-maps: Visually highlights "Hot Zones" in code where logic is overly complex or poorly structured.

III. Professor's Integrity Suite (Fingerprinting)
   - A. Logic Hashing: Strips variable names and maps the "shape" of the code to detect structural plagiarism.
   - B. Hardcoding Detection: Flags instances where students have "cheated" the logic by mapping specific inputs directly to outputs.

IV. Metric & Scoring Module
   Complexity Scoring: Provides an automated "Cleanliness Score" based on nesting depth and branch density.

V. Bulk Reporting Module

A. Batch Scanner: Points our tool at a folder of 100 assignments; it runs the analysis on all of them automatically.
B. Gradebook Export: Generates a CSV/Excel file summarizing the "Health" and "Integrity" of every student's submission.

7. Tools and Technologies

- Language: Java 21 (for modern pattern matching and performance).
- Core Libraries: * JavaParser: For Abstract Syntax Tree (AST) manipulation.
  - JGraphT: For mathematical graph theory and path analysis.
  - GraphStream: For interactive, real-time graph visualization.
- GUI: JavaFX (focusing on a clean, professional dashboard for evaluators).
- IDE: IntelliJ IDEA.

8. Expected Outcome

- Dramatically reduced grading time for our professors through automated "Spaghetti Code" detection.
- Higher Academic Integrity via structural logic comparison that goes beyond simple text-matching.
- Centralized Logic Data allowing evaluators to see where an entire class is struggling.
- Objective Feedback for students, as our system provides mathematical proof of poor design choices.

9. Future Enhancements

- AI-Based Predictive Grading
- Web-Based Evaluator Portal
- Cross-Language Support
- Integrated Telemetry for Lab Sessions
- IDE Plugin Integration

10.Conclusion

Logic-Flow Code Visualizer (LFCV) aims to provide a complete digital solution for academic code analysis and structural evaluation. By automating daily grading activities and enabling the visualization of trends, this system improves efficiency, accuracy, and decision-making capabilities for our professors. Ultimately, our project ensures a higher standard of healthcare for the "codebase," enhancing the quality of Computer Science education through intelligent data analysis.