Modified Functional Requirements

1. Changes to Functional Requirements (FRs)

1.1 Functional Requirements to Remove

The following FRs in Submission #1 are no longer relevant to the updated project scope and should be removed:

- **FR7–FR13**: Folder naming convention validation, pattern checking, and duplicate detection.
  Reason: The project requirements do not include enforcing a naming pattern or detecting duplicates; the system only needs to read subfolders and run tests.
- **FR20–FR22**: Delete Test Suites, Upload/Re-import Test Suites, and Secure persistent storage.
  Reason: Only one Test Suite is used. Uploading and deleting suites are not part of the real system.
- **FR23, FR25**: Detailed side-by-side expected vs actual output and logging for traceability.
  Reason: Version 2 results do not require detailed per-test output visualization—only pass/fail and success rates.
- **FR3–FR6**: Complex folder validation, correction prompts, and detailed error guidance.
  Reason: The program only needs to attempt compilation and record failure; advanced folder validation is unnecessary.

1.2 Modified Existing Functional Requirements

The following FRs remain, but their scope needs to be clarified to match Version 2:

- **FR14–FR16 (Compilation)**
  Modify to reflect that:
  - The system must detect the Java file containing public static void main(.
  - Compiling the main file compiles all other files automatically.
  - If compilation fails, the student receives "NO_COMPILE" as the success rate.
- **FR18–FR19 (Test Suite Creation & Editing)**
  Modify to indicate:
  - Only one Test Suite is needed for all runs (not multiple suites).
  - This suite must be reusable for both the first submission and the resubmission.
  -

1.3 New Functional Requirements to Add

FR-A1: Save Test Suite Results

The system shall allow the professor to save the results of running a test suite into a serialized result file that can be reloaded later.

FR-A2: Load Test Suite Results

The system shall allow the professor to reload a previously saved result file and display student success rates.

FR-A3: Execute Test Suite on a Second Submission Folder

The system shall allow the professor to run the same test suite on a different root folder representing the resubmission.

FR-A4: Compare Two Result Files

The system shall allow the professor to select two saved result files and display, for each student:

- The success rate from submission 1
- The success rate from submission 2
- OR a special code (e.g., "NO_RESUB", "NO_COMPILE")

FR-A5: Detect Main File Automatically

The system shall scan all .java files in a student's submission folder to find the file containing public static void main(.

FR-A6: Compile Multi-File Programs

The system shall compile the main Java file, automatically triggering compilation of all required files in the same folder.

2. Changes to Non-Functional Requirements (NFRs)

2.1 NFRs to Remove

The following NFRs no longer apply:

- **NFR3–NFR6**: Naming pattern flexibility, logging for auditing, duplicate handling.
- **NFR7–NFR8**: Secure logging of compilation with timestamps.
- **NFR11–NFR12**: Detailed output display formatting and secure storage of detailed logs.

These features exceed the project's intended complexity.

2.2 New/Updated NFRs

NFR-A1: Success Rate Format

Success rates shall be displayed strictly as a fraction:
number of test cases passed / total number of test cases.

NFR-A2: Special Codes for Missing or Failed Submissions

When comparing two result files, missing or non-compilable submissions shall display short codes (e.g., "NO_RESUB", "NO_COMPILE").

NFR-A3: Simplicity of Visualization

Loaded results and comparisons shall be displayed in a clear, text-only, non-editable interface.

3. Changes to Use Cases

3.1 New Use Cases to Add

Add three new use cases:

1. **UC: Save Test Suite Results**
   Actor selects a file path; system serializes the results.
2. **UC: Load Test Suite Results**
   Actor selects a results file; system reloads and displays stored results.
3. **UC: Compare Two Results Files**
   Actor selects two result files; system outputs success rates or special codes for each student.

3.2 Modifications to Existing Use Cases

UC: Execute Test Suite

Add the following clarifications:

- This use case may run on the **first submission** or **resubmission**, depending on the selected root folder.
- After execution, the use case **prompts the user to save the results**.

4. Changes to the Class Diagram

4.1 New Classes

1. TestSuiteResult

- Stores the complete results of one test suite run.

Relationships:

- TestSuiteResult is associated with exactly one TestSuite.
  (One TestSuite → Many TestSuiteResult objects, i.e., one result for submission 1, another for submission 2.)
- TestSuiteResult contains multiple StudentResult objects.
  (Each TestSuiteResult holds one StudentResult per student.)

2. StudentResult

Purpose: Represents the outcome of applying all test cases to a single student's program.

Relationships:

- Each StudentResult is associated with **one** student's program folder (represented by Program in Submission 2).
  (Program → 1 StudentResult per run)
- All StudentResult objects are **contained within** a single TestSuiteResult.
  (One-to-many: TestSuiteResult → StudentResult)

3. ResultFileManager

Purpose: Handles saving and loading serialized test-suite result data.

Relationships:

- Used by the **Coordinator** (the class responsible for managing execution flow).
  (Coordinator → uses → ResultFileManager)
- Handles reading/writing of the TestSuiteResult class.
  (ResultFileManager ↔ TestSuiteResult)

4. Add Class: ComparisonResult

Purpose: Represents the comparison outcome between two result files for a single student.

Relationships:

- Created and returned by the **Coordinator / Controller** class during comparison operations.
  (Coordinator → creates → ComparisonResult)
- Does **not** replace or modify StudentResult. It is a separate comparison output used only for display.

- .