

# Automated Unit Tests Assignment

**Name:** Diksha Kushwah

**Course:** Enterprise Software Engineering Practices

**Date:** 2nd Oct, 2025

**Repository:** <https://github.com/dikshakushwah04/esep-grade-calculator>

## Part 1: Requirements Clarification

From my discussion with Timothy, here are three items I would ask to clarify the details of things:

1. I would ask Timothy if a student with no grades in any category (assignments, tests, or essays) should receive a final grade, or if that category would be considered to have a zero.
2. I would ask if any extra assignments, tests, or essays that may be added beyond the regular list would adjust the weighted average calculation. Are those part of the average as they are a part of the calculation of the assigned grade, or are they purely optional?
3. I would ask if the final numerical grade should be rounded prior to becoming the letter grade. For example, if a student earns an 89.6, should that round to a 90 (A), or remain a 89 (B)?

## Part 2: Understanding a New Project

**Tech stack used:**

- The project is written in **Go (Golang)**.
- Uses Go's built-in testing framework (**testing**) for unit tests.
- Uses Go modules (**go.mod**) for dependency management.

**File summaries:**

1. **README.MD** – **Instructions** for **running** and **setting up** the project.
2. **grade\_calculator.go** – **Where** the GradeCalculator class and methods **for adding** grades **and getting** to a numerical **grade**, **and then converting that grade** to a letter **grade is implemented**.
3. **grade\_calculator\_test.go** – **Stores** unit tests to **verify** GetFinalGrade **gets the** correct letter **grading where appropriate** for **each situation**.
4. **go.mod** – Go module file the module name and Go version **is listed**.

**Tests in the repository:**

- Yes, the repository contains tests.
- They check if **GetFinalGrade** returns correct letter grades (A–F) based on input grades.
- To run tests: **go test -v**

**Resources used to understand the code:**

- Go official documentation (<https://golang.org/doc/>)
- Tutorials on unit testing in Go
- Exploring the GitHub repository files

#### **README update (for next user):**

- Added instructions for running tests and viewing coverage:

# Run all tests

```
go test -v
```

# Run tests with coverage

```
go test -cover -v
```

## **Part 3: Updating Tests**

#### **What I did:**

- When I initially tested, some of the tests failed because the code was buggy, specifically the `computeAverage` function was computing averages for essays incorrectly (it was looking at exams when calculating essays).
- I fixed the code in `grade_calculator.go` to calculate the average grades correctly for all assignments, exams, and essays.
- I also updated the tests to match the required weighted average and grade scale:
  - Assignments: 50%
  - Exams: 35%
  - Essays: 15%
- After updating, all tests passed.

#### **Grade scale used:**

- $A \geq 90$
- $B \geq 80$
- $C \geq 70$
- $D \geq 60$
- $F < 60$

## **Part 4: Adding Tests and Coverage**

**Code coverage after updates: 96.3%**

**Additional tests added:**

- I also tested it with letter grade C and D.
- I tested it for the case when zero grades are added (should return F).
- These tests helped ensure nearly all code paths are exercised.

**Refactoring suggestion to make testing easier:**

- I anticipate separating computeAverage to it's own helper function taking slice of the grades for haystack
- Also I would categorize all of the grades into one slice, with a Type field in order to simplify the averaging process, and to make future feature additional streamlined.

## **Part 5: Understanding Test Coverage**

**1. Can 100% code coverage be achieved without testing all cases?**

- No. Code coverage only indicates which lines have run in terms of testing the code. If we didn't test all scenarios, there may be logical paths which will never be exercised. So for example, there may be an if branch that would never run.

**2. Commenting out assertions:**

- If assertions are commented out, code coverage would still math, both lines were run but not verifying correctness in tests.

**3. Addressing this in a team setting:**

- I would recommend code review and test audit to ensure all cases have been tested, and that assertions are meaningful..
- Emphasize both coverage and test correctness.

## **Summary of Work Done in the Project**

1. I forked and cloned the repository.
2. I perused the code and learned how grades are added and calculated.
3. I fixed the bugs in grade\_calculator.go so that weighted averages computed correctly.
4. I also updated current tests and added new tests to cover missing situations.
5. I ran coverage and confirmed nearly all coded paths were tested.
6. I documented my findings, coverage, and suggested refactoring for future ease of testing.