MLA// OPTIONAL PORTFOLIO TASK

WEEK // SESSION	WEEK THREE // SESSION SIX
NAME OF SESSION	TDD WORKSHOP

ASSESSMENT FOCUS	TEST DRIVEN DESIGN APPROACH (TDD)

The following questions are designed to follow the two coding exercises included in the independent learning slide deck. Please ensure you have completed these exercises.

Theory Questions:

1. Explain the core principles of Test-Driven Development (TDD).
2. Describe the typical TDD workflow (Red-Green-Refactor).
3. How do requirements drive the TDD process?

Reflective Questions:

1. Reflect on your experience with the coding exercises in this lesson. Did you find TDD helpful or challenging? Why?
2. How did TDD influence your approach to coding? Did it change the way you think about writing code?
3. What are some potential challenges you foresee in implementing TDD in a real-world project?
4. How do you think TDD can contribute to the overall quality of a software project?
5. How will you use TDD in your MLA group project?

1. Core Principles of Test-Driven Development (TDD)

Test-Driven Development is built on several fundamental principles that guide the development process

Test-First Philosophy

Tests are written before the production code, to ensure that every piece of code has a specific purpose. The purpose of this approach is to force developers to think about the desired behaviour and interface before implementation. Tests serve as executable specifications that define what the code should do

Incremental Development

Development proceeds in small, manageable increments. Each cycle focuses on making one test pass at a time. This reduces complexity and makes debugging easier when issues arise

MLA// OPTIONAL PORTFOLIO TASK

Safety Net Creation

Tests provide immediate feedback when changes break existing functionality so developers can refactor with confidence, knowing tests will catch regressions. The test suite acts as living documentation of system behaviour

Requirements-Driven Design

Tests translate requirements into concrete, executable examples; each test represents a specific requirement or user story this ensures that all development effort directly supports business needs

Design Through Testing

Writing tests first influences code design toward better structure. TDD naturally encourages loose coupling and high cohesion, the need to test code forces consideration of dependencies and interfaces

2. The TDD Workflow: Red-Green-Refactor

The TDD cycle follows a distinct three-step process:

RED Phase: Write a Failing Test

Write the smallest possible test that captures the next piece of functionality

The test should fail initially because the feature doesn't exist yet

This ensures the test is actually testing something meaningful

Focus on what the code should do, not how it should do it

Example: Write a test for a calculator's add method before the method exists

GREEN Phase: Make the Test Pass

Write the minimal amount of production code needed to make the test pass

Don't worry about perfect design or optimization at this stage

The goal is to quickly achieve a passing state

Even "cheating" with hardcoded values is acceptable initially

Example: Implement the add method with the simplest logic that works

MLA// OPTIONAL PORTFOLIO TASK

REFACTOR Phase: Improve the Code

Clean up both test and production code while keeping tests passing

Eliminate duplication, improve naming, and enhance structure

Make design improvements now that behaviour is protected by tests

Ensure code follows good practices and patterns

Example: Extract common logic, improve variable names, optimise algorithms

Key Rules:

Never write production code without a failing test

Never write more test code than necessary to demonstrate a failure

Never write more production code than necessary to pass the test

3. How Requirements Drive the TDD Process

Requirements serve as the foundation and driving force of TDD:

Requirements as Test Cases

Each requirement is broken down into specific, testable scenarios

User stories are translated into concrete test cases

Acceptance criteria become the basis for test design

Behaviour-Driven Examples

Requirements provide examples of expected system behaviour

Tests capture these examples in executable form

Edge cases and error conditions from requirements become test scenarios

Focused Development

Only features specified in requirements get implemented

Prevents over-engineering and feature creep

Ensures development effort aligns with business value

MLA// OPTIONAL PORTFOLIO TASK

Progress Tracking

Passing tests indicate completion of specific requirements

Test coverage shows which requirements have been implemented

Failed tests highlight incomplete or incorrect implementations

Iterative Refinement

As requirements evolve, tests are updated to reflect changes

New requirements drive additional test cases

The test suite becomes a living specification of the system