LAB 3: Component Design & Implementation

1. OBJECTIVES

1.1 Discuss the system architecture design with your lab supervisor
1.2 Design subsystem
1.3 Implement subsystem

2. INTRODUCTION

2.1 In the last lab session, the team have worked together to decide on the candidate architecture and the requirements / responsibilities allocated to each subsystem. The provided and required interfaces of each subsystem are thus clarified.

2.2 In this lab, you will design and build the subsystem.

2.3 In the next lab, you will begin to integrate your subsystem with other subsystems, to perform a partial slice of the required system functionality.

2.4 Before start this lab, discuss with your lab supervisor for your overall architecture design, what are the quality attributes that you want to address, and how your design handles these quality attributes. Please prepare a 15 min presentation with 5 slides.

3. PROCEDURE

3.1 *Design subsystem*

3.1.1 Starting from the refined architecture diagram from Lab #2, identify entity, control and boundary classes in a Component Diagram.

3.1.2 Determine and delegate responsibilities to the various control classes, following the design principles of encapsulation, loose coupling and high cohesion.

3.2 *Implement subsystem*

3.2.1 Realise your subsystem design in code. There ought to be traceability from design to implementation.

3.2.2 Provide documentation for your code, especially in the subsystem provided / required interfaces.

3.2.3 Compile and unit test your code using white / black box techniques. Check your code (including test case code) into the repository.

3.2.4 Integrate the components of the subsystem following the usage scenarios / functionality described in the use cases. You may need to write test stubs / drivers to simulate the behaviour of the subsystems with which you will integrate in the next Lab.

4. DELIVERABLES

- Component Design
  - static model –Component Diagram
  - dynamic model – Communication Diagrams (1 or 2 based on some interesting scenarios in your system)
  - data persistence design e.g. ER Diagrams (if applicable)
- Component Implementation
  - code
  - documentation
  - test cases

5. REFERENCES

- JUnit - http://junit.org/
- Apache Subversion – http://subversion.apache.org/
- Object-Oriented Software Engineering: Using UML, Patterns and Java – B. Bruegge and A. Dutoit