

# CS 445 Software Engineering I

## Project Manual (v2.1)

Originally Created by:

Dr. Wee Wee Sim [wsim@semo.edu](mailto:wsim@semo.edu)

With edits by Joshua Schulz

### Overview

#### **1. INTRODUCTION**

Students in the CS445 Software Engineering I course will undertake team projects to solve real-world problem. They will solve the problem(s) with tools and practices commonly used in the industry. Students will traverse the entire Software Development Life Cycle (SDLC) to deliver a medium size project in a team. Students will contribute to different types of SDLC work products, e.g. requirements, design, code, test cases. The project is to be scoped such that it can be completed by a team of 4 (or 5) students within 15 weeks (Fall 2021 semester).

The “**Project Manual**” document provides students guidance about when to do what by setting requirements for each of your project deliverables. However, you must organize your own work and manage your own time in the project. The project manual will be updated as the semester proceeds. The class will be notified via announcements when there is a new updated version of the project manual.

#### **2. ROADMAP**

There are **5 project deliverables (PDs)** to be completed and submitted by each team during the course. The 5 project deliverables (PDs) are highlighted in Table 1.

**Table 1.** Project Deliverables

PD#	Focus	Deliverables
PD1	Requirements Elicitation (4 weeks, excluding week1)	<ul style="list-style-type: none"> <li>Functional Requirements</li> <li>Non-Functional Requirements</li> <li>Data Dictionary</li> <li>Initial User Interface (UI) Mockups<sup>1</sup></li> <li>Initial Use Case Model ○ Use Case Diagram ○ Use Case Descriptions</li> <li>Project Plan<sup>2</sup> ○ Work Breakdown Structure (WBS) ○ Gantt Chart</li> </ul> <p><sup>1</sup> Initial UI mockups may be created on paper and pencil via screenshots captured.</p> <p><sup>2</sup> Project Plan: WBS and Gantt Chat diagrams may be created using Microsoft Project or other project management tool of your choice.</p>
PD2	Requirements Analysis and Modeling (3 weeks)	<ul style="list-style-type: none"> <li>Complete Use Case Model</li> <li>Identification of major Entity Classes, Boundary Classes, and Control Classes</li> <li>Initial Class Diagram</li> <li>Dynamic Model ○ Sequence Diagrams (minimum 4 sequence diagrams from 4 main use cases) ○ State-Chart Diagram</li> </ul>
PD3	Design and Implementation (4 weeks)	<ul style="list-style-type: none"> <li>Complete Class Diagram</li> <li>Complete Dynamic Model</li> <li>Complete User Interface (UI) Design</li> <li>System Architecture</li> <li>Application Skeleton<sup>1</sup></li> </ul> <p><sup>1</sup> Skeleton (partial) code of your application.</p>
PD4	Implementation and Software Testing (3 weeks)	<ul style="list-style-type: none"> <li>Test Cases and Test Results ○ Black Box Testing ○ White Box Testing</li> <li>Working Application Prototype ○ Complete Source Code</li> </ul>

PD5	Final Project Report and Project Demo	<ul style="list-style-type: none"> <li>• Final Project Report Submission<sup>1</sup></li> <li>• Project Demonstration<sup>2</sup></li> <li>• Peer Evaluation</li> </ul> <p><sup>1</sup> Final Report is a SRS (Software Requirements Specification) document containing all the deliverables (PD1, PD2, PD3, and PD4) in a well-organized professional report. SRS document template (available on Canvas) should be used for the final report.</p> <p><sup>2</sup> Project demo: a 15-20-minute recorded video presentation of the working application prototype (e.g. mp4 format or uploaded onto YouTube)</p>
-----	---------------------------------------	--

## Project Description

### 1. INTRODUCTION

Students in the CS445 Software Engineering I course will work in teams of 4 (or 5) students to develop some working software applications to address a real-world business need. Students will put into practice concepts and skills learnt in this course as well as in the past related courses, in the context of a real-world problem. They will solve the given problem(s) with tools and practices commonly used in the industry. Students will traverse the entire Software Development Life Cycle (SDLC) to deliver a medium-size project in a team. Students will contribute to different types of SDLC work products e.g. requirements, design, code, and test cases.

Students **can choose any type of software applications** they would like to develop. Examples of software applications for the CS445 course are:

- eCommerce (e.g. video store, music store, book store, apparel)
- Restaurant management
- Travel guide
- Real estate
- Rideshare service (e.g. Uber, Lyft)
- Railway/train ticketing
- To-do list
- Health guide

The general requirements for the software application prototype to be developed for this course team project are the following:

- The application can be developed on desktop, mobile, or web platform.
- The application can be developed using any type of programming languages of your choice (e.g. Java, C++, C#, JavaScript)
- The final software application must be able to run (i.e. a working prototype) on desktop, mobile, or web platform.
- The final software application must be free of virus/malware.

- Students can use any UML modeling tools/IDEs to create different kinds of model as required in the deliverables. Examples of modeling/IDE tools are:
  - Visual Paradigm ○ Modelio ○ Umbrello UML Modeler
  - Microsoft Visio ○ Sparx Systems
  - MagicDraw UML
  - IBM Rational or Rational Rose
  - Altova Umodel ○ Netbeans IDE ○ Eclipse IDE ○ Visual Studio IDE ○ etc.

## **Project Deliverable PD1**

### **1. OBJECTIVES**

The following objectives are to be completed by the specified due date for PD1:

- 1.1. Meet your team members, discuss your individual strengths and interests and select (nominate) the team leader. These activities should be completed by the end of week 1).
- 1.2. Decide the software application to work on for the project, This activity should be completed by end of week 2.
- 1.3. Elicit and document software requirements: functional and non-functional requirements.
- 1.4. Develop data dictionary.
- 1.5. Develop initial Use Case Model.
- 1.6. Develop a mockup prototype of the User Interface (UI).
- 1.7. Develop project plan: Work Breakdown Structure (WBS) and Gantt Chart.

### **2. INTRODUCTION**

- 2.1. In this and the next 4 project deliverables, you will work as part of a team. Each team will consist of 4 (or 5) members.
- 2.2. All work products generated in the project deliverables should be updated on the team chosen repository. Examples of popular repositories include GitHub, Google Docs, Microsoft Teams.
- 2.3. During the requirements elicitation, the software requirements are determined and documented in appropriate technical format (a formal Software Requirements Specification (SRS) document is required as a final project report that documents all the project deliverables).
- 2.4. Use Case diagrams enable the proposed system to be easily visualized and help in refinement of the requirements.
- 2.5. User Interface (UI) mockups help in elicitation and refinement of the software requirements.

### 3. **PROCEDURE**

#### 3.1. **Meet your project team**

- 3.1.1. Your first task is to **meet** your project team members. Project teams have been pre-determined prior to the beginning of week 1. You can check your team members (including emails) from the People tab on the course page on Canvas. A list of the project teams will also be posted in Week 1 Module on the course page on Canvas.
- 3.1.2. **Choose a name** for your team. **Appoint a team leader** among your team members. It is the responsibility of the team leader to ensure that there is a fair distribution of work among the team members and that each member (including the team leader) is pulling his/her own weight.

#### 3.2. **Elicit and document requirements**

- 3.2.1. Determine the **target users** of your application. Elicit **functional** and **non-functional** requirements of the application. Note that you may wish to have some of the team members act as customers and the rest be the project development team. This is an artificial way of defining the requirements but within the constraints of this course project it is a practical way to proceed.
- 3.2.2. Document all the requirements in appropriate and **proper technical format** (i.e. proper requirement statements format. *Refer to instructor PPT slides in WK4 lecture*). The requirements should clearly state who performs what system functionality, taking what input and producing what output.
- 3.2.3. **Atomise** the requirements such that they are verifiable and traceable (*refer to instructor PPT slides in WK4 lecture on atomized requirements*). You will be writing test cases in project deliverable PD4 to verify the requirements. You will also have to demonstrate traceability from requirements to the final product.
- 3.2.4. Document important **terms** of your application (e.g., user, device, input, output, API,) in a **data dictionary**. Explain each term with a brief description. Identify attributes of each term and relationships between terms.

#### 3.3. **Visualize and refine requirements with use case model**

- 3.3.1. From the set of functional requirements, identify the preliminary **use cases**. Depict them on a **use case diagram** using the UML modeling tool.
- 3.3.2. Do not be over ambitious and make your project overly complicated by identifying too many use cases. **5-7 use cases** are sufficient for the project.
- 3.3.3. For each use case, start writing the **use case description** about how the user interacts with the system to carry out the system functionality. As a rule-of-thumb, each use case should have a maximum of 67 steps in its flow of events. A small (1-2 steps) use case indicates that the functionality has been sliced too finely; a large use case can be further broken down.
- 3.3.4. Iterate over your use case diagram to identify **included** use cases, **extended** use cases, and **generalization relationships**, if any (*refer to instructor PPT slides in WK5 lecture*).

3.3.5. A **Use Case Description template** is available to provide you a guide in creating the use case descriptions for your project (*the table on the last page of the template file*). You should use the Use Case Description template to create all your use cases. The template is posted in the Week 1 module on Canvas.

3.3.6. The initial Use Case Model will be refined and elaborated/updated as your application evolves during the course of the semester.

### 3.4. **Develop initial user interface (UI) mockups**

3.4.1. Develop a UI prototype by sketching a series of screens either using paper and pencil (or pen) or some suitable drawing tool. The UI design should incorporate good human-computer interaction (HCI) principles. The UI mockups do not need to be working software. They can be hand-drawn and screens captured.

### 3.5. **Create project plan**

3.5.1. Develop the project **Work Breakdown Structure (WBS)**, which is a visual and hierarchical decomposition of your project into manageable chunks. Your WBS shows all the project parts in an organized chart.

3.5.2. Create a **Gantt chart** from the Work Breakdown Structure (WBS) and track the tasks across time. Your Gantt chart should show the start and finish date of each task, their dependencies, and their relationship to each other in terms of sequencing. All the tasks identified in the WBS should be reflected in the Gantt chart.

3.5.3. There are a number of project tools you can use to create the WBS and Gantt Chart. One such tool is the **Microsoft Project**.

## 4. **DELIVERABLES**

- **Documentation of Functional and Non-Functional Requirements**
- **Data Dictionary**
- **Use Case Model** (initial) ◦ Use Case Diagram  
◦ Use Case Descriptions (using the Use Case Description template, *i.e. the table on the last page of the template file*)
- **User Interface (UI) Mockups** (initial) • **Project Plan** ◦  
Work Breakdown Structure (WBS) ◦ Gantt Chart

Submit all the required documents and diagrams on Canvas by **TBD**.

## **Project Deliverable PD2**

## 1. **OBJECTIVES**

The following objectives are to be completed by the specified due date for PD2:

- 1.1. Refine or correct Use Case Model created in PD1.
- 1.2. Build the Conceptual Model Class Diagram (identification of major Entity, Control, and Boundary classes).
- 1.3. Create an Initial Class Diagram.
- 1.3. Build the Dynamic Model (Sequence Diagrams, State-Chart Diagram).

## 2. **INTRODUCTION**

- 2.1. Starting from the use case descriptions created in PD1, more details about the system requirements can be uncovered through requirement analysis. By building a model of the application (problem) domain, the team can validate, correct and clarify the requirements.
- 2.2. Requirement elicitations and analysis are highly iterative activities. Expect to re-visit your use case diagram and user case descriptions to clarify and add detail, as you work through requirements analysis.
- 2.3. Using the methodology described in the supplementary material: *Bruegge & Dutoit* [BD] Chapter 5 (PDF chapter 5 will be posted on Canvas), your team will develop the **Analysis Model** consisting of (a) the Object Model, also known as the **Conceptual Model**, and (b) the **Dynamic Model**.
- 2.4. The Conceptual Model is usually depicted in the Class Diagram, while the Dynamic Model is usually depicted in the Sequence Diagrams and the State-Chart (or State-Machine) Diagram.
- 2.5. The Analysis Model forms the basis for system design and is to be refined in PD3.

## 3. **PROCEDURE**

### 3.1. **Refine Use Case Model**

- 3.1.1. **Finalize your Use Case Diagram** with clear definition of actors and use cases.
- 3.1.2. Keep **refining the Use Case Descriptions**. Describe clearly use case precondition, flow of events, and alternative flows (using the Use Case Description template – the table on the last page of the use case description template document).
- 3.1.3. Expect to re-visit your Use Case Diagram and User Case Descriptions to clarify and add details as you build the Conceptual Model and the Dynamic Model.

### 3.2. **Build the Conceptual Model**

- 3.2.1. From the Use Case Descriptions and the terms in the Data Dictionary in PD1, **identify the initial Entity, Boundary, and Control** objects.

- 3.2.2. From your understanding of the classes' responsibilities, apply the **dependency stereotypes** for Entity, Boundary or Control classes and create the **early Conceptual Model Class Diagram** (similar to HW6)
- 3.2.3. Create an **initial Class Diagram**, depict these classes and the **relationships** between them (i.e. identify appropriate association, generalization, aggregation).

### 3.3. **Buid the Dynamic Model**

- 3.3.1. For **each use case**, model the **interaction between the Boundary, Control and Entity classes** from the Conceptual Model Class Diagram, to enact the flow of events in the Use Case Description.
- 3.3.2. Specify the **messages** and **parameters** passed between classes in the **Sequence Diagrams**.
- 3.3.3. Model the system behavior in a **State-Machine Diagram** of UIs based on the UI mockups developed in PD1. This State-Machine diagram of UIs is also referred to as a Dialog Map of the system. Refine and further specify your user interface design.
- 3.3.4. Expect to move back and forth between the Conceptual Model Class Diagram and the Dynamic Model because studying interactions and behavior will reveal additional attributes, operations and relationships of classes.

## 4. **DELIVERABLES**

- **Complete (finalize) Use Case Diagram**
- **Complete all Use Case Descriptions (using the Use Case Descriptions template – the table on the last page of the use case descriptions template document)**
- **Conceptual Model Class Diagram depicting the key Entity, Boundary, and Control classes.**
- **Class Diagram** (initial) with attributes, operations (methods), and relationships between classes.
- **Dynamic Model** ○ Sequence Diagrams (minimum 4 sequence diagrams for PD2) ○ State-Chart Diagram

Submit all the required documents and diagrams on Canvas by **TBD**.

## **Project Deliverable PD3**

### 1. **OBJECTIVES**

The following objectives are to be completed by the specified due date for PD3:

- 1.1. Transform the Analysis Model into a Design Model.
- 1.2. Determine proper System Architecture.
- 1.3. Address key design issues and use appropriate design patterns.
- 1.4. Begin implementing your Design Model.

## 2. **INTRODUCTION**

- 2.1. During software design, the Analysis Model is transformed into the corresponding Design Model. In object-oriented methodology, the important classes in the Analysis Model can be mapped onto the Design Model.
- 2.2. The boundary between analysis and design stages in the SDLC is fuzzy. At the tail end of analysis, you were already beginning to think about some of the design issues listed below. You will **continue to refine the Sequence Diagrams and the State-Chart Diagram** from PD2.
- 2.3. Make design decisions to incorporate good principles such as **encapsulation**, **loose (low) coupling** and **high cohesion**. The use of appropriate **design patterns** will improve reusability, extensibility and maintainability.
- 2.4. Once your design model becomes stable, you can start **mapping your Design Model onto code**, i.e. start implementing your system.

If you implement your design using Java or other Object-Oriented languages,

- Each UML class is mapped onto a Java class.
  - The **generalization** relationship is mapped to an **extends** keyword.
  - The interface **realization** relationship is mapped onto an **implements** keyword.
  - An **x-to-many** association is mapped onto a **Collection**.
- 2.5. As you start the implementation, keep in mind the traceability from requirements to design, then to implementation and test cases.
  - 2.6. In addition to software tools provided by the CS lab, you can choose any technologies that you deem relevant to the design and implementation of your application. You need to **justify your choice** in the project demo and final documentation. Also, be sure that the software tools you use will work for you before committing to them.

## 3. **PROCEDURE**

### 3.1. **Transform Analysis Model into Design Model**

- 3.1.1. Modify your Conceptual Model from PD2 to make a draft **Design Model**.
- 3.1.2. Determine appropriate **System Architecture** suitable for your application. Be sure to provide **justifications** on the chosen system architecture of your application.

3.1.3. Apply **Design Patterns** where appropriate. *Refer to lecture materials from WK10–WK12.*

3.1.4. Re-visit your Sequence Diagrams and State-Chart Diagram from PD2. Add details e.g. **attributes** and **operations** (methods) to your class model (Class Diagram).

### 3.2. Address key design issues

3.2.1. In your design, address key design issues:

- Identifying and storing **persistent data**.
- Providing **access control**.

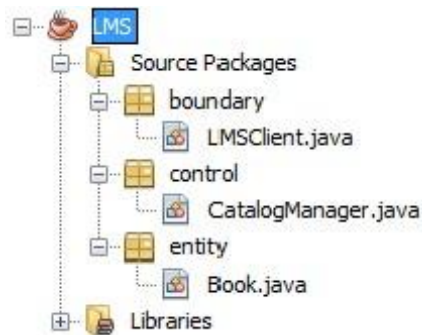
3.2.2. Apply **Design Patterns** where appropriate. *Refer to lecture materials from WK10–WK12.*

3.2.3. **Add details** to the Class Diagram and Sequence Diagrams of your Design Model by adding **attributes**, **operations** to your Class Diagram and **messages** and **parameters** passed between classes in the Sequence Diagrams.

### 3.3. Implement your design into code

3.3.1. From the Class Diagram, **generate the skeleton code** (partial code such as class name without body). Implement the behavior as documented in the Sequence Diagrams and State-Chart Diagram.

3.3.2. You may wish to organize the classes into packages or folders, following the stereotypes in the Design Model:



3.3.3. Document the classes and the key **public** methods to convey design intent and usage using Javadoc or similar techniques for other programming languages:

```
/**
 * This class implements the Book entity with
 * the attributes title, author, ISBN
 *
 * @author jane doe
 */
public class Book {
```

- 3.3.4. To facilitate team collaboration, upload the source code and libraries into an appropriate repository. Your team members can then check out the code. Examples of popular repositories include GitHub, Google Docs, Microsoft Teams.

#### 4. **DELIVERABLES**

- **Complete Design Model** ○ Class Diagram ○ Sequence Diagrams ○ State-Chart Diagram
- **System Architecture**
- **Application Skeleton Code**

Submit all the required documents and diagrams on Canvas by **TBD**.

## **Project Deliverable PD4**

### **1. OBJECTIVES**

The following objectives are to be completed by the specified due date for **PD5**:

- 1.1. Implement your application.
- 1.2. Design test cases using Black Box and White Box Testing Techniques.
- 1.3. Plan your project demo (2-15 minutes recorded video presentation).

### **2. INTRODUCTION**

- 2.1. The Design Model from PD3 must now be implemented in a working application prototype.
- 2.2. The initial application skeleton code in PD3 needs to be further refined and completed (i.e. some more code to be written) to accommodate additional behavior such as error handling and optimization.
- 2.3. The class' implementation in code should be traceable to its design and requirements.
- 2.4. As the coding gets underway, the **test cases** are developed in parallel. The inter-mingling of code-test activities helps build confidence as working code snippets are systematically built into a sizable piece of code.
- 2.5. The **implementation and testing phase of the Software Development Life Cycle should be completed by PD5** in which your team will complete a final project report and demonstrate your working application prototype. *Note that if you do not have time to implement all functionality of your design you can just implement and demonstrate the main functionality in the demo. Your design should be complete but there is no need to implement all the functionality if your schedule becomes tight.*
- 2.6. It is now time to prepare the product for delivery and presentation. As the time allocated to the project demo is limited to 15-20 minutes per team, prepare the materials and sequence of the demo carefully, so that the salient features of your application are highlighted in your video presentation.

### **3. PROCEDURE**

#### **3.1. Implement your application**

- 3.1.1. From the Design Class Diagram, generate the skeleton code. **Implement the behavior** (i.e. operations) as documented in the Sequence Diagrams and the State-Chart Diagram.
- 3.1.2. You may wish to organize the code into packages or folders, following the stereotypes in the design class model. *Refer to PD3 description.*
- 3.1.3. Document the classes and their key **public** methods to convey design intent and usage. *Refer to PD3 description.*

3.1.4. Use an appropriate repository to facilitate team collaboration.

### 3.2. ***Design test cases using Black Box and White Box Testing Techniques***

3.1.1. Design test cases using **Black Box** and **White Box Testing** techniques:

- **Black Box Testing:** test requirements and specifications. *Refer to PD1 functional requirements.*
- **White Box Testing:** test implementation details (source code).

3.1.2. Techniques to use for Black Box testing are: **Equivalence Class** and **Boundary Value Testing** techniques. Test requirements and specifications from **ONE important use case**. An example of an important use case could be an Account Registration or Account Login use case, i.e. specifications of username and password). *Refer to instructor PPT slides in WK13 lecture.*

3.1.3. Design **White Box Testing** test cases using **Basis Path Testing** technique. Test **TWO important methods from a Control Class** that implements some important application logic according to the requirements specification. *Refer to instructor PPT slides in WK14 lecture.*

3.1.4. Minimize the number of test cases through the careful selection of test input using proper Black Box and White Box Testing techniques learnt in the course.

3.1.5. Execute your test cases and document the testing results. Note that you can develop automatic test cases using proper automatic testing framework, for example, JUnit, or you can perform manual testing guided by your test cases.

3.1.6. Your test cases should include the following components (in table format):

- Test Case ID
- Test Case Description
- Tester
- Test Date
- Test Input
- Expected Output
- Actual Output

As an example:

Test Case ID		
Test Case Description		
Tester		
Test Date		
Test Input	Expected Output	Actual Output
---		

---		
---		
---		

### 3.3. **Plan for recorded video demo**

- 3.3.1. Craft the demo script and sequence so that you can maximize the opportunity to present your product to the client (i.e. me – the instructor).
- 3.3.2. The client (me – the instructor) is eager to see how well the various specified functionalities are carried out in your application prototype. Demonstrate the various usage scenarios by the different end-users.
- 3.3.3. In the presentation, highlight innovative solutions in the application prototype, as well as elements of good software engineering practices and system design.
- 3.3.4. Prepare/determine who in your team will present the presentation. Your team can send one representative for the presentation. Alternative, some or all team members can participate in the demo by playing different roles of actors. It is completely up to your team on how to run your demo.

## 4. **DELIVERABLES**

- **Test Cases and Testing Results**
  - Black Box Testing
  - White Box Testing

Submit document containing test cases and testing results on Canvas by **TBD**.

*Note: your team can include the test cases and testing results in the SRS document (for the final project report) OR submit a separate Word document containing the test cases and testing results, in addition to the SRS document (final project report)*

## **Project Deliverable PD5**

### **1. OBJECTIVES**

The following objectives are to be completed by the specified due date for PD5:

- 1.1. Complete and submit **PD4** deliverables (test cases and testing results).
- 1.2. Complete and submit final project report (SRS document).
- 1.3. Submit recorded video presentation.
- 1.4. Complete peer evaluation.

### **2. INTRODUCTION**

- 2.1. After 15 weeks of hard work, it is time to produce your final project report and showcase (demo) your team project! The marking rubrics for the final project report and demo will be made available to the class and posted at an appropriate time on Canvas.
- 2.2. You should carefully prepare your demo script. Make sure your product works “perfectly” (i.e. smoothly). You will be in control of how to start your demo, how to proceed, and how to wrap it up. Carefully plan and edit your recorded presentation to avoid all those pitfalls in your product, for example those actions that may crash your system.
- 2.3. During the demo, highlight all **additional** features your product can support that are not presented in the final project report. Also highlight all good Software Engineering concepts and designs you applied in your project.

### **3. PROCEDURE**

#### **3.1. Demonstrate your working application prototype**

- 3.1.1. Each team will have about 15-20 minutes for the demo. Some suggestions are provided in the “Project Recorded Video Presentation” section (page 18). However, it is completely up to your team how to run the demo.

#### **3.2. Submit Deliverables PD4 and PD5**

- 3.2.1. Submit **project deliverable PD4** on the PD5 submission link on Canvas. PD4 (test cases and testing results) may be included in the SRS document. Alternatively, your team may submit a separate Word (or PDF) document containing the test cases and the testing results.
- 3.2.2. Submit **final project report** on the PD5 submission link on Canvas. The final project report is the **SRS** document containing all the project deliverables properly organized (with a comprehensive Table of Contents) and a section on the **work distribution**: a list or table clearly indicates the members’ tasks/responsibilities (i.e. who did what). The SRS template is available on the course page on Canvas.

***NOTE: o Points will be deducted if the SRS document is not used to document all your deliverables.***

- *Points will be deducted if the Use Case template (the table on the last page of the Use Case template document) is not used to document your use case descriptions. Use Case template is available and is posted on the course page on Canvas.*

3.2.3. Submit application prototype implementation **source code**. The source code may be contained in a zip file or a folder and upload the zip file or the folder on the PD5 submission link on Canvas.

3.2.4. Submit **project recorded video presentation** (e.g. mp4 file) on the PD5 submission link on Canvas. If your demo video is too large to be uploaded on Canvas, your team may upload the video to YouTube and submit the valid YouTube URL link to your demo video.

### **3.3. Submit Peer Evaluation**

3.3.1. This project is a team work! Therefore, it is important to evaluate your team members on how well/much your team members have contributed to the final project product.

3.3.2. Each member of a team must submit his/her peer evaluation form (individually) in the Peer Evaluation submission link on Canvas.

3.3.3. EVERY team member needs to provide a **FAIR, UNBIASED**, and **CONFIDENTIAL** evaluation of his/her teammates' contributions to the project.

3.3.4. Peer evaluation form will be posted on Canvas at an appropriate time.

## **4. DELIVERABLES**

- **Project Deliverable PD4** ○ Test cases and testing results (Black Box Testing, White Box Testing)
- **Final Project Report (SRS document)**
- **A 15-20-minute Recorded Video Presentation**
- **Application Prototype Implementation Source Code**

*The above items are to be submitted as a **team**. You can include all the above items in a zip file (properly organized your items/folders) and submit the zip file on the PD5 submission link on Canvas.*

- **Peer Evaluation Form**

*Submit peer evaluation individually on the Peer Evaluation submission link on Canvas.*

Submit all the above required items on Canvas by **TBD**.

## **Project Recorded Video Presentation**

## 1. **INTRODUCTION**

- 1.1. In addition to the project final report, each team will prepare and submit a 15-20-minute video demonstration/presentation of the working application prototype.

## 2. **SUGGESTIONS/TIPS FOR DEMONSTRATION/PRESENTATION**

- 2.1. You can have one member of your team to represent your team for the presentation. Alternatively, some or all team members can participate in the presentation by playing different roles. Your team has the freedom on how/who to present.
- 2.2. You can provide voice-over to explain the various functionalities/features, together with PowerPoint slides. If your application also applies to mobile platform, you can video capture the application on the mobile phone. You are free to choose how you want to present your working prototype. Important thing to note is that the functionalities/features and the user interface (UI) of your application must match with the information documented in your SRS document.
- 2.3. Your application must be working prototype. Present/demo the working functionalities/features of your application (imagine that you are presenting your working prototype for your client) and make sure that your functionalities/features work.
- 2.4. Your team does not have to present/demo all the functionalities of your application. Remember, that you are only given **15-20 minutes** to present/demo your working application. Therefore, carefully select a few important or interesting functionalities/features you want to showcase to your client (i.e. in this case, the instructor who will be grading/assessing your project—me!). Below are some of the things you should consider to present/demo in your recorded video:
  - A short **introduction/description** of your application.
  - The **technologies** used for your application.
  - An overall **User Interface** (UI) of your application.
  - The **Use Case Diagram**, highlighting the important/main/interesting use cases (or functionalities/features).
  - Short explanation of the PD3 deliverables; specifically, the **Conceptual Model**, **Class Diagram**, **Sequence Diagrams** (demo one or two sequence diagrams), and the **Architecture Diagram**. Again, you are only given 15-20 minutes time to present. Therefore, you do not need to explain every single component in these diagrams; select a few important/interesting components to showcase in your recorded video.
  - Discuss some **good Software Engineering practices** you applied in the course of your project, an overview of your system design, and how these good practices and designs allow you to easily **support envisioned further upgrades**. You may consider making a short PowerPoint presentation for such highlights.
  - Demonstrate the good **traceability** in your project deliverables. Your team may choose one or two specific use cases, and present the relevant class diagram and sequence diagrams for the chosen use case (i.e. how the use cases relate to the class diagram and sequence diagrams).
  - It is also recommended that your team mentions some **good designs** (i.e. design patterns, if any) you applied and how you implemented them.

- Present the **test cases and test results**.
  - For Black Box testing, highlight the test requirements and specifications from the one important use case (submitted in PD4) and demonstrating how you apply the **Equivalence Class** and **Boundary Value** testing techniques.
  - For White Box testing, present one of the two methods that you submitted in your PD4 deliverables, demonstrating how White Box testing on the two methods was performed.
- Conclusion/Final thoughts.

Mention some interesting things about your project, e.g. **future work/enhancements**, **areas to improve on**, etc.

As you can see from the above, there are several things to present/demo about your project. Therefore, careful planning and practice on the demo are necessary.