Software Engineering
Semester Project

This is a pdf version of all the material that is provided on Canvas. Please be sure to look over everything and ask if you have any questions.

Project Description

As opposed to computer science where a significant amount of focus is on algorithms and computational modeling, software engineering focuses on the creation and development of software systems. There are many aspects to taking an idea and turning it into a completed product. Furthermore, most of these software products are created and engineered as a team effort. For these reasons, you will be completing a semester long project.

The project is designed to give you an overview of the major stages of a software project as well as to help you develop basic skills in working with teams. Software Engineering has shifted considerably over the years, and there are multiple processes and methodologies out there. For this course, we will be using a modified agile approach that completes a project over multiple iterations. However, it is important to note that different organizations will adopt different practices and processes that meet their needs.

Project Deliverables and Schedule

This includes the deliverables desired for this semester's project along with due dates. Failing to participate in any phase will result in automatically failing the project (receiving a zero for the project).

Due: September 13th

> ***Group Contract*** – Outlines the vision for the project, users, and features, as well as describes the problem that the software system is trying to solve and elicits buy in from team members. Additionally, outlines disciplinary policies for failing to contribute.
> A template is provided on Canvas.

Due: October 4th

> ***Product Backlog*** – Identifies major functionality and requirements that need to be met before the product is considered complete.
> ***Scenarios*** – For each functionality or use case for your system, you should provide a scenario description. Attach each scenario to its associated card in the product backlog. A template is provided on eLearning.
> ***Join Version Control and Issue Tracking***

Due: October 25th, November 15th, December 6th

> ***Software Requirements and Design Document*** – This document should be developed alongside the project. It breaks down important details for the software projects, including the material from the product backlog along with descriptions of the system's and each feature's design. A template is provided on eLearning.

Demonstration Video – Each group should create a short video that demonstrates the progress of the group in the project. Each video should demonstrate new functionality and discuss any changes from the previous version. The video will be 5-10 minutes in length and should be uploaded to YouTube with a link to the videos given in the Software Requirements and Design Document.
Worklog – Worklogs are discussed in the course syllabus. Each group only submits one. Make sure to keep these updated.
Peer Reviews – Peer Reviews will be completed as an online assignment in eLearning.
Source Code and Tests – These will be available as a branch within GitHub.

Due: Last Week of Classes (Before Finals)
Final Presentations – Includes a high-level overview of the software system and a brief demonstration of the working product. *Slides should be submitted to me by December 6th.*

Breakdown of Project Grading

Grading for the project is broken down into the following categories and allotments:

• Group Contracts	7.5%
• Product Backlogs and Scenarios	17.5%
• Software Requirements and Design Documentation	30%
• Demonstrations, Development, and Testing	45%

The presentation is graded separately and is worth 5% of the course's final overall grade. You will find rubrics for each of the above on Canvas.

Assessment of Individual Contribution

These projects are meant to be completed in a team and not by an individual carrying the weight of the team. For this reason, there are multiple checks built into the project to help measure individual contribution. Contribution works as a scalar multiplier applied to the group's grade for an individual deliverable. This multiplier ranges from 0 to 1, where 0 represents that you provided minimal support to your team and 1 represents that you were a productive member of the team. As an example, suppose that your team earned an 84 for requirements. Members with a 1 for contribution would receive the full grade. However, suppose that you had a .5 for your individual contribution. This means that you would only receive a 42 as your grade with the remaining 42 points being lost. Individual contribution is primarily determined through two different sources:

- Each individual will sign the group contract.
- Each individual is responsible for updating the group's worklog as they complete work throughout the current portion of the project. All logs are submitted to the group dropbox for that portion of the project.
- Each individual is responsible for completing peer reviews. Failure to complete peer reviews will result in a 0 for contribution as you didn't know your teammates well enough to review their contribution.

These measures are put in place to help ensure that no one person is carrying the entire workload of the group. That is not fair to that individual. At the same time, I am not looking for hero coders. You will not get extra credit for trying to tackle the project alone. If you work with your team and contribute to their success, you should have no problems with individual contribution.

Project Ideas

These are only suggestions. Your group may choose a different project. However, if you choose a project not on the list, you must get approval for the idea before submitting your vision and scope document.

- ***Visualization of algorithms***: this project requires the team to produce a graphical visualization of different algorithms. The user should be able to pick the algorithm to display from a predefined list, and be allowed to enter the initial inputs for the algorithm to operate on. Multiple algorithms should be available, and intermediate results should be displayed to the user. Examples of possible algorithms include graph algorithms, sorting algorithms, tree traversals, network routing algorithms, etc. The user should be able to save a particular configuration so that they can refer back to it at a later time. This involves understanding how to create dynamic changes in a graphical user interface.

- ***Railway Passenger Reservation System*** – Develop a system that allows passengers to reserve seats on trains, and for railway administrators to manage trains and tickets. The following are a list of features that you must account for:
  - Tickets are valid for one to seven days based on distance traveled: 250 km/2 days, 500 km / 3 days, 1000 km/ 4 days, 2000 km/ 6 days.
  - There are four classes of tickets: hard-seat, hard-sleeper, soft-seat, soft-sleeper. Not all of these are available for every train.
    - Hard-seat: Coach, typically loudest and most crowded travel
    - Hard-sleeper: sleeper carriage with doorless compartments
    - Soft-seat: Coach with reclining seats
    - Soft-sleeper: Luxury class, four bunks in closed compartments
  - Tickets should detail the train, the car, and the berth.
  - Dining cars will be present on some trains. Passengers can choose what meals that wish to pay for before boarding.
  - Trains can pass through multiple cities and may stop at different points along the way. Express trains do not stop at any intermediary stops.
  - Prices are based on the class of service and length of the journey. Surcharges exists for express service.

You will need to be able to store any changes made to the above data so that it is not lost when the program shuts down. You are to use your imagination in implementing a system that supports these requirements, and develop other requirements you feel are important. Feel free to add detail to the situation.

- ***Campus Parking Management System***: Develop a system that manages college parking. Your group is to develop an easy to use system which supports most or all of these features:
    - Maintain a catalog of employees and students, cars, parking areas and assignments.
    - Track parking lot/space usage patterns.
    - Track violations, fines, etc.
    - Issue parking stickers / hangers with lifetimes ranging from hours to a full 12 months.
    - Assess appropriate charges for parking authorizations.
    - Track parking regulations, assignments and location and status of lots (e.g., under repair, percentage full, etc.).

  You will need to be able to store any changes made to the above data so that it is not lost when the program shuts down. You are to use your imagination in implementing a system that supports these requirements, and develop other requirements you feel are important. Feel free to add detail to the situation.

- ***Ticket Allocation & Sales***: Develop a software system that manages athletic ticketing operations. Fans can be classified as students, faculty, university staff, alumni, trustees, public officials, celebrities and the general public. Also, the athletes and the athletic staff want to have tickets for special guests. Some fans want season tickets, and some want select game packages. There are various categories of seats and boxes in the larger venues. Parking lot spots at the venues are also part of packages offered to fans willing to pay with more desirable spots costing more and shuttle bus service an option.

  It will have to:
    - Maintain all the information about schedules, venues, fans, sales packages, etc.
    - Provide for sales for all the ticket packages the athletic sales group chooses to offer.
    - Provide for browsing available seat locations.
    - Enable system managers to set aside blocks of seats for special groups or promotions.

  You will need to be able to store any changes made to the above data so that it is not lost when the program shuts down. You are to use your imagination in elaborating these and in formulating new requirements. Feel free to add detail to the situation.

- ***Board and Card Games -*** at a minimum, all games should be able to detect and deny illegal moves made by a human opponent (suggesting possible moves after repeat incidents), allow for play against both a human and a computer player, and detect and gracefully handle detection of end-of-game states and winning/losing.
  - ***Chess***
  - ***Texas Hold'em Poker***
  - ***Euchre***

  When recommending moves for Chess, you should avoid moves that will result in immediate capture (assuming system is not intelligent enough to determine intelligent sacrifices). Players should be able to save their game and return at a later time.

- ***Physics Simulator*** – Build a program that demonstrates aspects of rigid body mechanics. This project requires the team to produce a graphical visualization of different physics laws. The user should be able to setup a model for a scene in which different types of objects are either launched or dropped through the scene. Users can add obstacles, surfaces, etc. with different properties to see how the presence of these new items affects the object's movement. The user should be able to save their scene. This involves an understanding of basic physics concepts and how to create dynamic changes in a graphical user interface.

- ***Tactical RPG*** – Your group will build a game similar to games such as Final Fantasy Tactics and Disgaea. These types of games are turn-based strategy games that involve the player using a team of heroes to defeat an opponent's team. Your game would need to provide the following:

  - Multiple levels with each level increasing in difficulty. This can be done by increasing the number of enemies (within reason), changing the composition of the opposing team, or using stronger opponents.
  - Multiple types of enemies – it gets boring to fight the same thing repeatedly.
  - Different classes – the player should be able to create teams of different types of characters and to have such character progress through new classes after completing objectives
  - Usable items – you can choose to either give a player a set number of items at the beginning of each battle, provide an in-game shop that uses currency won from each battle, or have the characters earn items during the game. Items can include health restoring items, items that restore ability points (e.g., mana), or status restoring items.
- ***Adventure Game*** – Players will explore an unknown region, making choices that affect the game along the way. The game should include the following:

  - Dialogs between the player and other characters along the way. Dialogs should be decision driven, meaning players have the opportunity to choose multiple topics for discussion.
  - Items that the player can collect and make use of throughout the story.
  - Story points that force the player to make decisions. Each decision has different consequences.
  - Interaction points that enable the player to interact with other objects in the world.
  - A navigable world.

- ***Serious Game*** – Serious games are games that have a purpose outside of pure entertainment. An example of such a game is providing a scenario that an employee of an organization may have to deal with in order to improve their training and development. This can be done as part of one of the previous two ideas, or as a separate idea. The game should include the following:
  - A realistic simulation or experience that reflects the goal of the game
  - Decision points and interactions that require the player to demonstrate the skills needed to complete the simulation. These should require the player to complete actions and not just make simple text-based choices.

I reserve the right to make changes to the requirements or design of a project at any point throughout the semester.