CPSC:480 Software Engineering Project 2

Part 1: Project selection

Select a software development project for consideration meeting these criteria:

- You must be able to define at least ten functional requirements of the software. If a project has substantially more than ten functional requirements, consider narrowing the scope to focus on a sub-portion of the proposed software that could still have value.
- The software must be constructable (so no software for time travel)
- You should plan to use this product for projects 2, 3, and 4; while change is possible, it may significantly increase the amount of work that needs to be done.
- You will not need to construct a complete, working product during this class; you should, however, expect to contribute *some* code toward development of this product.
- A project that you *are* able to complete as a team during the semester will provide the most value to you in terms of both portfolio content and software engineering skills mastery.

Steps:

- Each team member should propose one software project, including the project goal, scope, and definition, along with ten functional requirements presented as user stories. Use the exercise 2 assignment as a template. This may be a new (greenfield) software project, a significant enhancement to a previous project, or a contribution to open-source software.
- As a team, select one of the proposed projects to move forward with. You may modify the proposed functional requirements at this point, and must add at least 5 non-functional requirements. Prioritize all requirements from highest to lowest. Also select a team name. Submit the proposed team name, project, and list of prioritized requirements through Brightspace for approval by **Monday, Sept 26, 11:59 PM**. I will create a GitHub repo for the team and provide access to team members to submit the rest of the work.

Part 2: Requirements engineering

- The online diagram editor at https://draw.io is recommended for creating diagrams. Students should also have access to Microsoft Visio through the University's Office 365 subscription. Do not use a simple bitmap editor like Microsoft Paint as these images are difficult to modify as requirements evolve. Diagrams do not need to use the formal UML standard but should clearly and accurately convey the relevant information. See examples in lecture slides or textbook.
- From your proposed project, develop a class model capturing the relevant object types, their attributes and operations, and relationships.
- Construct a scenario model (sequence/swimlane diagram) for each functional requirement. Each team member must construct at least one scenario model, but the team as a whole is responsible for completion and accuracy.
- Construct at least one behavioral model (state diagram) modeling part of your system.
- Write a full specification of the software product, including the following. You may find a template online; the book recommends web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc:
    - The artifacts of part 1
    - All of the diagrams
    - A paragraph explaining why this project has value
    - Any operating environment requirements and assumptions
    - Interfaces for users and other system
    - Anything else relevant to the project.
- Content should all be posted to GitHub, including source materials for all diagrams.

Checkpoint: Due Sunday, Oct 2, 11:59 PM

GitHub repo should contain:

- Each student's alternative proposal that was considered.
- Source files for all diagrams.
- Project specification document.
- At least two commits or pull requests from each team member.
- A release containing the pdf version of the specification.

Checkpoint grades will be 80% shared and 20% individual, as follows:

- 10% - Original project proposal and requirements from each user (individual)
- 10% - Sequence diagram contribution & 2 commits or pull requests from each user (individual)
- 05% - Proposal submission and prioritized requirements
- 10% - Class diagram
- 20% - Sequence diagrams
- 10% - State diagram(s)
- 30% - Specification
- 05% - Pdf release and directions followed

Part 3: Project Planning

Select the highest-priority requirements that can and should be met first. For each selected requirement, create a set of product backlog tasks indicating work to be done to meet the requirement. Each team member should complete task definition for at least one requirement, and the group should then meet to review and agree on task definitions. For simplicity, you may omit design tasks and focus on implementation tasks. For each task, capture the following information as a GitHub issue:

- Title
- Description of the task.
- Definition of done (i.e. how can another team member verify that the work is complete)
- Pre-requisite tasks (mention the other task to cross-reference)
- Priority: Critical (failure to complete task promptly puts project at risk), high (necessary to meet high-priority requirements), medium (useful toward high-priority requirements or necessary for moderate-priority items), low (not essential)

When requirements have been analyzed, meet to discuss tasks identified and produce estimates for each. Estimate the number of hours needed to complete the task. The recommended way to do this is for each team member to privately estimate the work required and take the median. If there is significant variation, discuss the reasoning. A designated team member should take thorough notes on this meeting content.

Part 4: Product Design

- Define the high-level design of the software: what language(s), what file/class solution structure, what build process, how the software will be executed and used, what needs to be done to begin meeting requirements. TODO – full definition coming soon

Part 5: Sprint planning and commencement

- Select the highest-priority tasks that can be completed in two weeks at 5 hours per week per team member based on projected estimates (10 estimated hours per developer). Identify who would complete each task (tasks do **not** need to all be actually completed, although you will need to do some work for at least one task) and assign the issue to that team member. You may do this in a meeting or over a chosen communication platform.
- Each team member should take their assigned tasks and identify what they need in order to complete each task (knowledge, code, tools, additional requirement detail), as if they were going to complete it. Record this information in a comment on the GitHub issue for the task.
- Hold a "standup" meeting (virtual or in-person, but everyone must attend) after everyone has reviewed their assigned tasks, and have each team member describe initial obstacles they are facing in completing the task and what they would expect to have done by the "next" standup meeting. A designated team member should take thorough notes on the meeting content.
- Each team member should make at least one commit related to one of their assigned tasks.
- Produce a brief report describing the state of the project as of the due date. It is fine if additional, unstructured progress is made, but work should be linked to a GitHub issue for the task. Do this in markdown on GitHub.

Submission

GitHub repo should contain:

- Issue for each task included in the sprint with title, body, assignee, project, and prereqs linked
- Comments for each assigned issue describing developer's needs.
- Product design schematics
- Notes on project planning meeting and standup meeting.
- At least one commit from each developer related to an assigned task.
- Markdown report on current status.
- Release containing final version of design doc and standup notes in pdf format.

Grades

Grading will be assessed as follows. 70% of the final project grade is shared and 30% is individual:

- 25% - Individualized grade on project checkpoint
    - 80% shared component of checkpoint grade = 20% of final project grade
    - 20% individual component of checkpoint grade = 5% of final project grade
- 10% - Task definition (individual)
- 05% - Identification of needs/obstacles, standup contribution, and issue comments (individual)
- 10% - Work item progress and tracking (individual)
- 20% - Design
- 10% - Sprint plan
- 10% - Meeting notes
- 05% - Report
- 05% - Directions followed

Participation

In the handouts folder of the class GitHub is a template called "Team Participation Survey". Each team member should submit a copy with their name by **Monday, Oct 3, by 11:59 PM** in physical form or via Brightspace. This is required and will count as a quiz. Additionally, you may submit a partial form, anonymous or not, at any point if you would like to raise any concerns. The purpose of this form is to ensure that all team members are meeting expectations and credit is being assigned appropriately.

Students may not receive full credit for team grades if they meet the following criteria:

- Average of contribution or expectations score from peers of 1.9 or less.
- GitHub contribution history and/or meeting reports do not reflect appropriate participation.
- No justification for poor performance or excuse from course requirements is provided

Conversely, students may receive up to 5% extra credit if they meet the following criteria:

- Average contribution or expectations score from peers of 4.1 or higher.
- GitHub history and/or meeting reports reflect outstanding skill and team commitment.
- Demonstrably fulfilled another team member's obligations for benefit of the team as a whole.

Notes

- Specialization of labor is an important component of any software engineering team. It is recommended to designate unofficial/semi-official responsibilities along the following lines. All team members will need to participate in work that does not fall into one of these roles, but dividing primary responsibility for different tasks will help ensure all members are able to contribute:
    - Project manager – Ensuring that project requirements are being met and contributions are on time; meeting schedules and agendas; communication.
    - Product owner – Overseeing product requirements, models, task definition, and non-architectural design elements.
    - Lead engineer – Overseeing technical product design elements and architecture, maintaining codebase (though it may be minimal), merging conflicting commits.
    - Technical writer – Composing specification text, meeting notes, and final status report.
    - Generalist – Flexible time allocation to assist with other tasks as needed.
- It is recommended to exchange phone numbers and establish a group chat in a platform of choice for ease of team communication.
- A successful project schedule might look like the following. Deadlines in bold, off-days in Italics):

	9/19 Read project assgnmnt	**9/20 Submit team requests**	9/21 Exchange contact info	9/22 *Complete exercise 2*	9/23 Develop idea and reqrmnts for it	9/24 Meet; Select project for proposal
9/25 Adjust and prioritize proposal reqrmnts.	**9/26 Makeup any delay & submit**	9/27 Review feedback & GitHub	9/28 *Complete exercise 3*	9/29 Develop class model	9/30 Develop scenario models	10/1 Meet; finish spec sections remaining
10/2 Makeup any delay & submit	10/3 *Study for midterm*	10/4 *Study for midterm*	10/5 Review feedback; assign task definition	10/6 Develop tasks for reqrmnts	10/7 Draft product design	10/8 Meet; esti-mate tasks & review design
10/9 Finalize design; asgn tasks	10/10 Review tasks;enter comments	10/11 Standup meeting	10/12 Make any progress on tasks	10/13 *Makeup for any delay*	10/14 Meet; Review status	10/15 Complete any work remaining
10/16 Makeup any delay & submit	**10/17 Complete teammate survey**	10/18 Review feedback	**10/19 Begin project 3**			