Object-Oriented Software Engineering Term Projects Spring 2018

Instructors: Uğur Doğrusöz and Eray Tüzün

As the course project, you are to develop a software system to solve a problem as described below. You are to work in groups of 4 (3-5) people as assigned, and complete your projects in two iterations/rounds.
The main objectives of the project are to gain experience in doing teamwork through all phases of software engineering and practicing object-oriented software development tools and techniques.
You are to implement your system in Java (unless all group members agree on using C++ or some other programming language with reasonable support for object-oriented programming) and extensively make use of GitHub project hosting platform for maintaining all there is to your project from source code to documents to issues.
The internal and exposed documentation of the system are essential.

Analysis

Analysis Document of a project, also known as a Software Requirements Specification (SRS), is produced as a result of the analysis of the system to be developed. The requirements provided by the customer are analyzed carefully and this document is produced as a result of a thorough analysis of the system at hand. In a sense, this document is a contract between the developer ("you") and the user/customer ("us" in this case).
Once you feel your analysis document is complete, you may begin your design. This phase is perhaps the most crucial of all and should be dedicated sufficient time. A project without a proper design might actually receive a failing grade even if its implementation is completely finished.
There can be many different ways in which to organize an Analysis document; the key point is being able to convey the model produced as a result of the analysis as clearly and completely as possible. One example organization for an object-oriented software system is as follows :

1. Introduction
2. Overview
3. Functional requirements
   3.1 Additional requirements (iteration 2 only)
4. Nonfunctional requirements
   4.1 Additional requirements (iteration 2 only)
5. System models
   5.1. Use case model
   5.2. Dynamic models
   5.3. Object and class model
   5.4. User interface - navigational paths and screen mock-ups
6. *Improvement summary (iteration 2 only)*
7. Glossary & references

Design

Design Document is mainly a non-exposed, internal document, and is a means for communicating ideas between different parts of a development team during the design phase as well as for proper maintenance of a software system. Considering the fact that people responsible for maintenance might not be the same people who have designed and implemented the software, it should be self-explanatory and complete. This

means just about anything used during analysis and design such as reasoning and rationale of key design decisions, design procedures (e.g. top-down decomposition of the system and specific architectural style used), and UML diagrams should be part of this document. Proper commenting of source code, of course, is an essential part of internal documentation.
Again there is no single way to organize such a report, a sample follows:

1. Introduction
   1.1 Purpose of the system
   1.2 Design goals
2. High-level software architecture
   2.1 Subsystem decomposition
   2.2 Hardware/software mapping
   2.3 Persistent data management
   2.4 Access control and security
   2.5 Boundary conditions
3. Subsystem services
4. Low-level design
   4.1 Object design trade-offs
   4.2 Final object design
   4.3 Packages
   4.4 Class Interfaces
5. *Improvement summary (iteration 2 only)*
6. Glossary & references

Implementation

Final Report is a document that discusses how the implementation went and whether or not any major changes to the design had to be made due to complications during the implementation. This report is to include some Exposed Documentation of the software system (could be an Appendix), corresponding to a User's Guide, and should provide everything that is needed to install and use the software properly. The document should also state the status of the implementation (i.e. any parts that are not implemented or are incomplete).
Again there is no single way to organize such a report, a sample follows:

1. Introduction: State where you stand at the moment with the implementation
2. Design changes: Describe any changes in the design and present high and/or level level design diagrams if needed
3. Lessons learnt: Tell us about your experience
4. User's Guide
   4.1 System requirements & installation
   4.2 How to use

Project Descriptions (A Rough Requirements Document)

We will let you decide your own projects but you need to let us know what particular project you have chosen, by the specified date. The project descriptions can be short and the details of the systems are left to your desire and imagination. Discussion with the TA or the instructor about your project is highly recommended.
Whenever possible, we ask you to avoid having to learn new technologies, complex algorithms and alike. For example, we discourage you to develop games with complicated networking needs, since having to learn networking can be quite time-consuming and would have little contribution to your OO software development skills. Similarly, a project with an advanced Artificial Intelligence or Graphics component is not recommended. Since this is your first project, where you should apply high and low level architectural styles

and design patterns, you should not use a library or a framework (e.g. a physics or game engine) that forces a particular design on your project.
If allowed by the owner, you may use ideas, images, screens, etc. from existing implementations but you must credit them by properly citing them. However, you are not allowed to lift source code from any such implementation! Should you need to make re-use of existing methods or relatively short code segments (cut-and-paste) longer than a few lines or make use of a 3rd party library, you must first get our consent!

Iterative Development

You are going to perform your development activities in 2 rounds. The first iteration will cover most basic requirements of your software using techniques and tools learnt thus far. The second and final iteration is to:

- Improve any problems in your design and implementation
- Add more advanced use cases (**required**)
- Use all techniques and tools learnt in this course

Use of Tools

You are required to use GitHub and UML **during all phases** of software development. In fact, we will **extensively make use of GitHub history to evaluate individual contributions**.
You are to create an additional branch for your project named “unstable”, and perform your regular documentation and coding activities on this branch. Please make sure to have a directory named “doc” containing your reports and other kind of documentation, and a directory named “src” containing your source code, which in turn should be organized with respect to your architecture / high level components.
When you’re ready to wrap up an iteration, you are to merge what you have on unstable branch to the master (default) branch and prepare a “release” (version 1 and version 2) from the master branch before the deadline for that iteration. Each such release will be considered as the final product of that iteration, and evaluation will be based on these releases as opposed to what’s currently in master or unstable branch.

Working as a Group

In order for a successful teamwork, regular meetings, especially during the initial phases of the development including design, are extremely important. Later on during the development, however, the tasks could be split for each person to work individually on. For a better organization and effective meetings, you are **required to keep meeting logs** (see this template)
You are strongly encouraged to express any serious concerns you might have in working with other(s) in your group to your instructor. And please do not wait till the end of the semester! Remember that each individual is equally responsible for all deliverables (documents, code, etc.)

Grading

Your grades will be based on your individual performance (which will be also dependent on a mandatory assessment of your performance by others and yourself) as well as the overall quality of the software developed. We suggest you evaluate yourself and your friends based on their contribution to the project and how cooperative they have been in a team environment (e.g. how easy it was to work with them and communicate ideas with them). **The peer grades submitted to the instructor will be kept strictly confidential!** The difficulty of the task undertaken will play a role in grading as well (you should be careful in the amount of tasks you undertake when you write your analysis report; try not to over or under-estimate the amount of effort needed to get full credit). We will give you feedback on this based on your first iteration results.

Once again, a project without a proper design might actually receive a failing grade even if its implementation is completely finished. Similarly, a project analyzed and designed properly but not implemented completely will likely have a passing grade.
We realize it is difficult to work in a group environment to get things done. We also realize that it's more difficult to measure the contribution and performance of individuals in such an environment. However, such exercises are crucial in preparing you for real life projects and we'll be as helpful as possible for the success of the projects and as generous as possible in grading.

Submission and Deadlines

You are to host and submit all there is to your project and conduct your development through GitHub as instructed. All milestones and due dates are as provided on the course Web page.
Once again the major objective of this project is to have you gain experience in doing teamwork; so try to be a team player, and have a positive and constructive attitude towards your teammates; one must learn to tolerate the differences in every aspect of life for success. Good luck and feel free to ask us any questions you might have regarding the project or the object-oriented software development concepts in general.