A Template for A Final Project in Software Engineering Application Design Document[1]

Last revised on December 9, 2018

[1]Derived from a previous document authored by *Dr. Eliezer Kaplansky.* and Dr. Mayer Goldberg

Contents

Guidelines

There are many ways to approach the task of modeling software systems. This document describes one possible way. It is a general suggestion to get you started, and should not be considered a set of rigid requirements. For example, projects that are very data-centric may consider following the order of subtasks, as presented in the following chapters, while more control-centric projects might find it more useful to focus more on defining behaviors and events, addressing data only at a later stage of the modeling process.

Chapter 1

Use Cases

Update, revise and extend the use-case scenarios in the application requirements document (ARD) to provide a comprehensive list.

Chapter 2

System Architecture

Describe the main components of your system architecture: What different software components are involved, where and how are they deployed, what functionality is delegated to each of these components. Make sure that you specify the target machine of all the files on your system ("*Does the file* `foo.dat` *live on the server or on the client?*"). You may use deployment diagrams to illustrate how the various system components interact.

Chapter 3

Data Model

This chapter describes the main information data domain of the application. Objects are real-world entities that have counterparts within the system. Associated with each object is a set of attributes and functions. Associating the functions will be handled in Chapter 5. The remainder of this chapter is devoted to analyzing the attributes.

3.1 Description of Data Objects

Of the many methods for developing a data model, there are two which two seem to stand out:

- Start by thinking about the *entities* you will represent, manage and manipulate in your system. These entities should translate into data objects in your model.
- Start by thinking about the *values* you will represent, manage and manipulate in your system. Find commonalities among them and try to find sensible groupings. These groupings should be the basis on which you define the data objects in your model.

Describe the main data objects. For each data object, include a description of its main attributes.

You may use any modeling language you like for the data model. For example, a class diagram that depicts the main instance- and class-variables (you should omit, at this point, any mention of methods). However you choose to represent your model, make sure it is readable and easy to understand.

3.2 Data Objects Relationships

Describe the relationships among data objects using class diagrams. For this chapter, model only the main relationships.

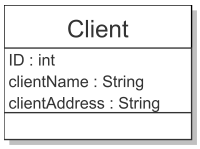

Figure 3.1: An example of a data object

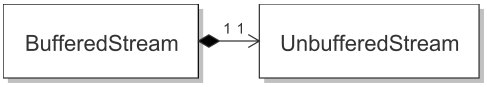

Figure 3.2: An example of two classes & the relation

3.3 Databases

If your application does not make use of a relational database, parts of this section are irrelevant to your project. Only include the parts that make sense for your work.

- Describe the data objects and the relationships among them in terms of Entity-Relation Diagrams (ERD).
- Describe the main tables, their structure and the types of the various fields.
- Describe the main transactions, which database entities (tables, collections, subgraphs, etc.) they modify, and how.
- Go back to the data object model and mark objects that map directly to database entities as such. *Do not perform this step if none of your data objects map directly to database entities!*

Chapter 4

Behavioral Analysis

This chapter describes the control flow of your system.

4.1 Sequence Diagrams

Analyze each use case, obtain a corresponding sequence diagram, and decide what methods are required by which classes, in order to support the given sequence diagram.

4.2 Events

Some software systems are best described and understood in terms of the events that control the behavior of these systems, i.e., what "happens", and how the system reacts in response. If your system fits this category, then detail and describe the events that govern the behavior of your system.

> **Example Scenario** Upon receiving a shutdown signal, the program should write a message to the system logger, save all open buffers, close all open files, including user files, system files, and configuration files, and terminate.

4.3 States

Some software systems are best described and understood in terms of a state-machine formalism such as *state charts*. If your system fits this category, then detail and describe the relevant states.

> Most systems have a few trivial states, such as `logged in / logged out`, or `online / offline`. Specify the main states of your system even if full state/events analysis for the entire system is not appropriate.

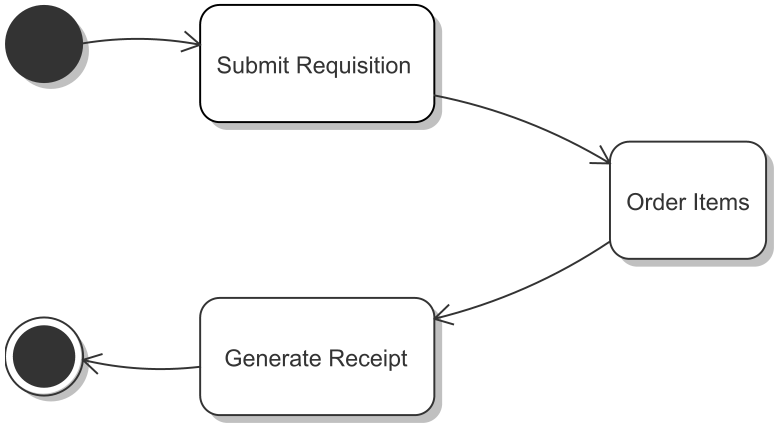

Figure 4.1: Handing a purchase request — An example of a state-driven logic

Chapter 5

Object-Oriented Analysis

Combine your data and behavioral analysis into classes, abstract classes, interfaces, and packages. Make use of *design patterns* to improve the structure of your code and the quality of your design.

5.1 Class Diagrams

Describe the classes in your system. For each class, mention attributes, operations, and relations to other classes.

5.2 Class Description

For the main classes described in Section 5.1, provide

- A textual description of its responsibilities
- For the main methods, list any invariants, pre-conditions, post-conditions (e.g., `ID` is never 0, $age > 0$, etc)
- Any implementation hints (e.g., *method* `foo` *should be declared* `synchronized`)

You may want to use the Object-Constraint Language (OCL) wherever possible, in order to keep the description concise and precise.

5.3 Packages

Any large software system will consist of many classes, which are best organized into a hierarchy of packages. Describe the package hierarchy of your software system.

5.4 Unit Testing

Design unit tests for each class. When designing the tests, try not to think about actual input and output values and concrete functions, but rather about properties of your classes and methods and their invariants.

Go over Section 5.2 and make sure you test the invariants, pre-conditions, post-conditions of your methods. Make sure you test any boundary conditions.

Chapter 6

User Interface Draft

Describe the main user interfaces of your system. Prepare simple drawings of the main screens. Keep the discussion at the level of inputs & outputs, rather than going into aesthetic details.

Chapter 7

Testing

Describe how you plan on testing the entire system. For each of the non-functional constraints listed in the ARD, describe how you intend to test that your software meets that constraint. Describe your actual tests. For each subsystem, describe the tests for this subsystem, the steps to conduct each test, the expected results and the actual results obtained.