A Template for
A Final Project in Software Engineering
Application Design Document*

Mayer Goldberg†

Last revised on September 8, 2005

Abstract

This document is the second of three documents describing the final project in the software engineering program. Specifically, this document describes the design for the software project. Please consult the supplementary document *Protocol On Written Reports* for detailed guidelines on how to use this template.

Contents

*Derived from a previous document authored by *Mr Eliezer Kaplansky.*

†Dr Mayer Goldberg, Office 58:312, Department of Computer Science, Ben Gurion University, Beer Sheva 84105. Office phone: (08)647-7873. Email: gmayer@cs.bgu.ac.il.

Guidelines

1 Use Cases

Update, revise and extend the use-case scenarios in the ARD to provide a comprehensive list.

2 Data Model

This section describes the main information data domain of the PD. 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 Section 4. The remainder of this section is devoted to analysing the attributes.

2.1 Description of Data Objects

Describe the main data objects, or business objects, that are to be managed and/or manipulated by the software system. For each data object, include a description of its main attributes – of those attributes of which the client should be aware. You could use, for example, a class diagram that depicts the main instance- and class-variables. You should omit, at this point, any mention of methods.

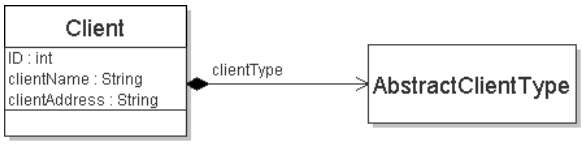

Figure 1: An example of a main class

2.2 Data Objects Relationships

Describe the relationships among data objects using class diagrams. For this document, diagram only the main relationships.

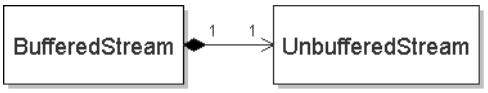

Figure 2: An example of two classes & the relation

2.3 Databases

If your application does not make use of a relational database, mark this section as ***Not Applicable***. Otherwise:

- 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 tables they modify, and how.

3 Behavioral Analysis

This section describes the control flow of your system.

3.1 Sequence Diagrams

Analyse 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.

3.2 Events

Some software systems are best described and understood in terms of the events that control the behaviour 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 behaviour 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.

3.3 States

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

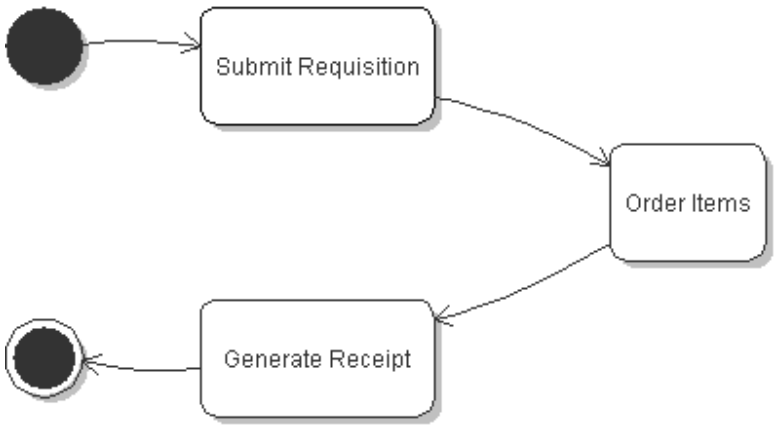

Figure 3: Handing a purchase request — An example of a state-driven logic

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.

4 Object-Oriented Analysis

Combine your data and behavioural 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.

4.1 Class Diagrams

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

4.2 Class Description

For the main classes described in Section 4.1, 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.

4.3 Packages

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

4.4 Unit Testing

Design unit tests for each class. Go over Section 4.2 and make sure you test the invariants, pre-conditions, post-conditions of your methods. Make sure you test any boundry conditions.

5 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.

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 æsthetic details.

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.

8 Task List

The culmination of your design is a comprehensive task list. Provide a hierarchical listing (i.e., an outline, or a tree) of the tasks involved in this software project. Top-level entries should be tasks that can be completed within two weeks. The entries at the leaves should be tasks that can be completed in one working day or less. For each task in your list, provide the following information:

- Task ID
- Brief title

- An estimate of the starting date
- An estimate of the duration of the task
- Description

While you should aim for the list to be as comprehensive and as complete as possible, changes to the tasks will be inevitable. You will need to keep the task list up-to-date by noting the following information:

- Actual starting date (don't delete your estimate; studying the difference between your estimate and "reality" will be helpful and informative)
- Actual duration of the task (again, don't delete your estimate!)
- Any updates. For example, during implementation, you may find that the task was too large and had to be broken down into simpler sub-tasks, or you may have to replace the task by one or more tasks, or else the task might become irrelevant and have to be deleted. In this case, note these changes, and refer to any task IDs that supersede the given task.

It may be most convenient for you to use a boilerplate form per task, such as the following:

Task ID			
Title			
Est Starting Date		Actual Starting Date	
Est Finish Date		Actual Finish Date	
Description			
☐ See sub-tasks			
☐ Superceded by tasks			

Since you are using MS Project throughout this course, you may as well use it to generate the tasklist.