Design Document

Version X.Y – yyyy.mm.dd

Created yyyy.mm.dd

Project Name
Name 1 (role-if any)
Name 2(role-if any)
…
Name N(role-if any)

Table of Contents

1 Introduction

1.1 Document Scope

Here you should describe what the purpose of this document is, and what it is not (e.g. it does not describe how a user uses the system.) No-one likes to have to read a few pages to figure out if they're looking at the right document.

1.2 Intended audience

This section should describe the stakeholders you are targeting this document at. This is another way to let people quickly discover whether they're reading the wrong document. Hint: in your current assignment, this is not the end-user.

1.3 Project Overview

What is it that you're building? Cross references or links to other documents may be relevant. This is a non-technical overview to provide the right context for understanding the following technical sections.

2 High-Level Design

2.1 Goals and Guidelines

You may adopt either of these two formats:

- Describe the goals and guiding principles ("philosophy") used to design the software, and that affect the architecture and high-level design. Describe the reason for their desirability (if possible, in relation to requirements), unless they are trivially obvious.
- Identify the software requirements and objectives that have a significant impact on the architecture, for example as a bullet list of single line items.

2.2 Architectural Strategies

Describe any design decisions and/or strategies that affect the overall organization of the system and its higher-level structures. How did you approach complicated parts of the design? Did requirements suggest paying special attention to reuse, unattended operation, a certain style or model of user interface, or performance vs memory? UML System Use Case diagrams constituting a high-level use case view may be useful to explain or justify your approach, but are not mandatory. Show only Use Case diagrams which depict significant, central functionality of the final system, or that have an architectural impact. Examples:

- Database. How is data storage management and persistence handled?
- Communication mechanisms, for example between components
- Expansion. Are there future plans for extending or enhancing the software? Will you leave interfaces for future expansion, 3rd party additions, etc.?
- Data Management. Are data items being transferred between subsystems or imported into/exported from your application tagged with a format version number in case future releases of the application change the transmission format?

- Error management. Should your software stop if there is an error, to avoid making the problem worse, or should it try to keep on running (perhaps skipping the affected task)?

These are not necessarily things that you should do, but it gives an example of the type of "design decisions" that belong in this section. Each decision should be accompanied by the reason it was taken.

2.3 High-Level Component View

The idea here is to describe the architecture, that is, the logical divisions that you have made in your project (GUI, backend logic, data access). In other words, how were the functionality and responsibilities of the system partitioned and then assigned to subsystems or components? A high-level UML Component diagram would be appropriate, without details such as the classes that implement them and without files; but do include interfaces (which you can revise in phase 2). You should describe the purposes (roles) that the components fulfill. Numbered subsections should be used for subsystems that need their own UML diagrams. You can also use numbered subsections for a textual description of the components. (i.e. "2.3.3 The ABC Component". Don't go into too much detail about the individual components themselves (there is a subsequent section for detailed component descriptions). Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of architectural design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them. If there are any UML Activity diagrams that had a particular bearing on the architecture, they may be included here; otherwise they should go into the detailed design section (Phase 2).

2.4 High-Level Deployment View

Use a UML Deployment diagram to describe any logical divisions that you have made in your project for deployment (e.g. server, client). As with all views, you should also provide a description and rationale excepted for the trivially obvious.

3 Detailed Design

3.1 Logical View

Here you will talk about the Classes which make up your design, using UML Class and Class package diagrams as appropriate. You should cover the reason that each class exists (i.e. its role in its package; for complex cases, refer to a detailed component view.) To help in this description, and to explain complex relationships between classes, you may use UML Object diagrams to show a snapshot of instances with values and their relationships (as examples). Use numbered subsections below (i.e. "3.1.3 The ABC Package".) Describe tactics such as abstracting out a generic DatabaseInterface class, so that changing the database from MySQL to Oracle or PostGreSQL is simply a matter of rewriting the DatabaseInterface class. Note that there isn't necessarily a one-to-one correspondence between packages and components.

3.2 Detailed Component View

Now that you have defined classes, you may provide details of how they help implement the components you defined previously, and the files that they would use. You do not have to define all components this way; use it for complex cases (for example, when the correspondence between packages and components isn't obvious).

3.3 Process View

Here is where you tie it all together. Give a description of how the objects interact to achieve the required results. Link these results to requirements. If several System Requirements are similar, discuss them together (as a group) to avoid redundancy. Use Sequence, State Machine or Communication Diagrams as needed. Any small differences from the illustration by a similar System Requirement in the group can be addressed in the text (group by numbered subsections).

3.4 User Interface Flow Model

The user interface is the application, from the point of view of the users. Do your classes and their interactions (the logical and process views) impose restrictions on the user interface? Would removing some of these restrictions improve the user interface? Use some form of user interface flow model as described in the slides to provide an overview of the UI steps and flows. Don't go into too much refinement. You should bring sketches of UI flow models to the video conference at the end of phase 1, or be ready to sketch some during the conference.

4 Appendix

Add any additional info you would like to refer to here.

5 Glossary

Build a sorted list of definitions here. This includes acronyms such as UM for Users' Manual, etc.

6 Revisions to this Document

Template Revised by Pascal Meunier, September 15, 2008. Rewritten in terms of 4+1 Views, UML 2.0, and User Interface flow models (with some inspiration from Agile Methods)

Second Revision September 18, 2008. Moved references to specific classes in an example of database abstraction from the Architectural Strategies section to the Detailed Design section (Logical View). Added examples to the Architectural Strategies section, and the suggested (optional) usage of Use Case diagrams.