Computer Science Capstone Design

Assignment: Software Design Document

Introduction

After having spent much of CS476 specifically trying to avoid committing to "implementation specifics" too early in the requirements specification phase, we are now ready to get to work on actually implementing the product. It's time to design some code!

The design specification is a description of your project's overall architectural design as well as the design of each major module. It is essentially the blueprint for your final product. If it is complete and well-executed, you should be able (in theory, at least) to send this document to any software development outfit and – without having anything more to go on – they would be able to realize your blueprint. As many software companies found out the hard way during the brief "software outsourcing to India" craze early in the last decade, reality is a little different: even with a very detailed software design specification, there are many tacit design details in the "context" of the design that are hard to impart to an outside team. In any case, a strong software design document serves to force you to make your implementation plans very explicit, and thereby often exposes problems in your envisioned architecture in plenty of time to think them through…rather than discovering them the hard way as you would if you had just started hacking away. Essentially the "blueprint" metaphor is pretty much right on: just as a blueprint allows others (colleagues, builders, structural engineers, the building department) to understand and evaluate your intended construction of a house, so the Software Design document allows careful review and evaluation of your implementation plan.

Ensure that you update this document as your project progresses, so that it remains a high-fidelity representation of your actual design. In fact, you can utilize most of this document again in the "As-built report", that is deliverable as the final project report, and that documents the final software architecture and use/maintain it.

This assignment write-up assumes that you will review your software engineering materials for details on how to clearly specify architectural and module designs; the content outline focuses on what should appear, not on how to actually create and present it.

Software Design Document Content Outline:

Cover Page

The customary cover page with the document title, your team's name and logo, team members, sponsor and mentor names, and the date. The document title should be annotated with some indication of which revision of the document this is (for example, "Version 1" or "Version 2.3").

Table of Contents

The contents of the document, and the page number on which each section begins.

Introduction (about 1-2 pages)

As always, every document you produce has to efficiently and clearly introduce a naïve reader to your project. You can recyle the intros that you've evolved last term for your other documents, just somewhat edited/extended/re-focused to serve as an intro to a Software Design document. After generally motivating and introducing the project and your planned solution, you'll want to focus add in a brief overview of the key user level requirements, the functional/performance requirements that you distilled from those, and any environmental constraints that you need to observe. Thus, the second half of this section presents the key pieces of your Requirements in condensed form.

A successful introduction will leave the reader feeling that the project is interesting, necessary/useful, and that the solution vision is exciting. More practically, the reader will also have a solid view of the requirements and constraints that this implementation must meet. This sets the stage for this document by providing a framework within which to understand and evaluation the architectural specifications that are coming up.

Implementation Overview (about a page)

Before you dive into real architectural details, you want to paint a big picture of what you have in mind and the technologies that you'll be relying on. Start by referring again to your solution vision, i.e., what it is that you will be building to solve the client's problem (a smart embedded robot controller, a cloud-hosted Web2.0 application the scales easily and flexibly, whatever). Then go ahead and introduce the overall approach you're taking: are you using a producer-consumer pattern, what software frameworks or packages you'll be using and what each of them contributes, etc. It's like introducing the names and roles of the player on your team before the real game starts. This just gives the reader a general sense for the tools and techniques you've chosen for getting the job done.

Architectural Overview (about 2 pages)

Now it's time to get into some high-level detail about how the system will actually be built. This section should consist of two parts: an architectural diagram of your system's high-level architecture, and a discussion of this architecture. The architectural diagram should focus on the most important of the system's components, while the discussion should explain: (a) the key responsibilities and features of each component, (b) the main communication mechanisms and information/control flows of the architecture, and (c) the influences from one or more architectural styles embodied by this architecture.

The Architectural Overview should leave the reader feeling like they have a pretty good overall understanding of what your system is doing, computationally speaking, to produce the desired behavior. It provides the introduction to the really nitty-gritty description of software components that is the meat of this document.

Module and Interface Descriptions (about 6-8 pages)

For each module in your architecture, provide a detailed design description consisting of the following information: (a) a short natural-language description of the responsibilities of the component and how it fits within the larger context of the architecture, (b) a UML class diagram of the classes involved in this component, or -- if your system does not adopt an object-oriented design -- a diagram of the sub-programs that the component consists of and how they are related, and (c) a description of the public interface of the component that explicitly outlines the services that the component provides -- for an object-oriented component, this will necessitate you elaborate on the

public methods (along with the return types and parameters) for the classes that this component consists of.

Implementation Plan (about 1 to 2 pages)
Provide a design-centric implementation timeline for your project; in other words, a schedule that focuses on the implementation of each component. Include a graphical display, such as a Gantt chart, that shows when you plan on completing work on the implementation of each module, its testing, and its integration with other components. Given that you have multiple team members, it would be surprising if you didn't have substantial parallelism in your chart, with multiple modules being worked on at any given time. In the narrative that goes with your Gantt chart, briefly walk us through the major development phases indicated in the figure and give any additional detail on scheduling or non-visible plan details. If possible, find a way to convey assignments of work to individual team members, either in the Gantt chart itself, in some sort of simple table presented subsequently, or in the descriptive narrative.

Conclusion (about half-1 page)
Every document has to have a conclusion…preferably a happy one. Having been buried in the architectural details, the job of your conclusion is to remind the reader of the big picture value and importance of the project, to summarize what was discussed in this document, and then say something about how it contributes positively to the project outcome.

DELIVERABLES

A complete software design document *draft*, professionally presented in hardcopy to your CS faculty mentor on or before the date shown in BBlearn. Since this is a draft, you don't need to bind it; staple is okay.

A final hardcopy document, professionally presented in hardcopy to your CS faculty mentor on or before the date shown in BBlearn.

Important Notes:

- The draft document is the first deliverable, and will be graded with *exactly the similar completeness/quality expectations as the final document*, but for fewer points. The final deliverable will be re-graded using the same expectations, but for 100 pts.
- **You must include your marked up draft** document to your CS faculty sponsor when you submit the final document; part of your final grade depends on how well you addresses critiques from the draft review.