Computer Science Capstone Design

Assignment: Final "As-Built" Report

Introduction

The final report summarizes "the story" of the project into a single coherent narrative. If someone had time to read just one document to understand everything about this project, starting from motivations and vision, summarizing the development process, and describing the resulting product, this would be the document to read. Much of the material to be included in the report can and should be shamelessly re-used from previous document products you've produced...with appropriate condensing, seaming together and cleaning up, of course. Be sure to update information adapted from previous deliverables! This is the final "as-built" report, meaning that it shouldn't contain "requirements as we understood them in October" or "Architecture as we thought it would be when we wrote our Design document in January"; you'll want to update/refine all such information so that it *reflects how the product actually turned out*...i.e., "as-built".

Target Audience: The central goal of Capstone is to provide the client with a "strong beta prototype" on the target software, to provide a strong basis for further refinement and extension into a final, deployable product. With this in mind, the final report should be targeted very consciously towards efficiently getting a future software team up to speed. Imagine that you have no clear idea of the project, but have just been assigned to pick it up and continue coding. What would be critical for you to know to get traction fast? You'd need an intro section that explained the big picture motivations clearly; you'd need a concise summary of requirements acquisition and the requirements that came out of it; you'd need an overview of the architecture along with some rationale for *why* it was designed this way; you'd need a discussion of implementation and key decisions (why this framework vs. others, etc.), and so on. The content outline below is driven by this focused need to get a future software team up to speed. In terms of tone and level of detail, this means you should focus on writing it *for a reasonably technically gifted audience*; don't be afraid to explain some technical detail.

If you write this document correctly, a future team will be able to read it, set up their programming environments and toolchain, and continue your work in short order.

Don't forget the cardinal rule of good technical writing: Coherent "story" and flow of narrative is everything! Your document should tell the story of your project from beginning to end, without abrupt transitions anywhere. Pay special attention to boundaries between sections (provide strong segues from one to next) and to section in which you present lots of detail (give section intro paragraphs to explain where you are in the story, what details you'll be presenting in the section, and

how they fit into the story). Reading your document should be a guided tour...not a jumble of unconnected sections and random dumps of detail!

Document length: As usual, there can be no specific minimum or maximum length here. Your aim is to concisely yet completely describe your project from motivations to design rational to internal architecture – at a level of clarity and detail that will allow a future team to quickly pick up where you left off. Grading will be based on this metric, i.e.: Was it clear? Was it complete? How effective is it as an on-boarding document? I'd say roughly 10-20 pages for most projects...

Content Outline

As usual, the detailed structure and content of the report might vary somewhat depending on the specific project it is for but, in general, one would expect to see the following sections.

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.

Table of Contents

The contents of the document, and the page number on which each section begins. Just index the main sections; don't index and give page numbers for every two-paragraph sub-sub-section you have in there!

Introduction

Once again, you'll need to efficiently and clearly introduce your project to readers with unknown levels of technical expertise. Assume that the reader has never heard about your project before, so you'll want to start with the big picture and motivations: the context of your system, including how it is to be used and by what kinds of users, while not forgetting any impacts of your work on organizations and how they operate. Then get more specific about the problems that your client has and what you built to solve them. You have practiced and refined similar introductions for previous documents during the project, so you should be quite good at this by now!

A successful introduction will leave the reader feeling that the project is interesting, necessary/useful, and that the solution vision is compelling and should clearly solve the client issues that were outlined. This sets the stage for this document by providing a framework within which to understand and evaluation the upcoming discussion of various project details.

Process Overview

Provide an overview of the process you used in developing your project. You should include information on the overall lifecycle development process you used as well as tools and artifacts that supported you in the use of this process, e.g., version control, CASE tools, task managers, etc. Describe the specific roles that each of your team members adopted, and what other team procedures you used to organize your work (recall the Team Standards deliverable).

Requirements
This section should summarize your functional and non-functional requirements (including qualities), as stated in the requirements document. Start by briefly reviewing the acquisition process, then summarize the requirements that resulted.

Architecture and Implementation
This section is the true heart of your document, and essentially comprises the "as-built" report for the project. The goal is very simple: to describe the software architecture and implementation of your system in detail sufficient to allow an incoming software engineer to read it once, and then immediately be productive in extending/maintaining your product. Ask yourself: if you came onto a new project and someone handed you an as-built document and said "read this and come in ready to work on it tomorrow", what you want to find in that document.

This section will generally consist of two parts. First, to give an overview of the system, you'll want to develop one ore more well-crafted architectural diagram of your system's high-level architecture, which you then walk the reader through, carefully discussing all of the components shown including: (a) the key responsibilities and features of each component, (b) the main communication mechanisms and information/control flows of the architecture, and (c) some illustrative example(s) of what the component does during a typical use case. Wrap up this overview part by outlining influences from one or more architectural styles embodied by this architecture.

In the latter subsections of this section, you will then give more detail on each of the major components. You don't have to go into ultra-low-level detail here. Keep the goal in mind: what information would you want about that component if you were an engineer new to the project?

Finally, in the concluding paragraphs of the section, you should discuss how what you built differs from what you intended to build: in other words, the differences between your prescriptive and descriptive architectures...between "as-planned" and "as-built". Example dimensions that could be applicable: What functions did you end up not building and why? What functions did you implement in a different way than what you planned and why? What architectural decisions did you have to change during development, and what drove these changes?

Testing
For this section, discuss your testing activities. This should include your overall testing strategy and the kinds of tests you ran to validate your implementation. Your aim here is give the reader an honest and solid overview of how the delivered software has been tested. Note that source-code tests are just one aspect of testing. Integration and usability tests should also be used to make sure that the functionality needed has actually been implemented in a way that users can access. This section should also describe the results of your testing and any changes you made to your design or source-code in response to testing results.

Project Timeline
Now that we know *what* your team produced, give a quick review of the overall project timeline. You could do this as a Gantt chart...or as a sequential list of milestones. In either case, briefly walk through the schedule you present, describing the key phases as well as any interesting factoids (how work was divided up in the team, who led on what, etc.) related to individual phases. Be sure to say where you are at this moment in the schedule your have outlined.

Future Work
Every good project will generate new ideas – by the dev team and/or the client – for things the product could or should do, but which were not originally envisioned within the scope of the project. Less desirable, there might also be features that *were* in the original scope of the project, but were not implemented for some reason.

Introduce, motivate, and discuss any important or promising features that you see as desirable and are recommending be considered for development by your client, e.g., in developing version 2.0 of your product.

Conclusion
As always, every document needs a conclusion that wraps up the discussion. Start by briefly reviewing the context and motivations of the project, your clients business work flow and what's wrong with it. Then state what you have built that addresses these needs, pausing to bullet out a few of the best specific features. Wrap up with discussing the impact your project solution will have (a) for your client, e.g., time-saved, accuracy improved, throughput increased...whatever. Put some specific numbers to it if you can. Then talk about (b) broader impacts (if any) that your product might have, i.e., other contexts or clients that might be able to use it, potential for broader markets, and so on. Close with some reflective comments on your team, the project process, and the Capstone class.

Glossary
If you use any terms that have special meaning (domain-specific terminology, for example), lay out the definitions here.

Appendix A: Development Environment and Toolchain (absolutely required)
Remember how uninformed you were on practical mechanics of how to actually develop code in your chosen environment when you started? How long it took you to figure out your toolchain and development cycle? One of the key pieces of "organizational knowledge" for a project is exactly this: how should a new team member configure their development machine, and what is the process leading from code to production of a runnable product? This is what you want to explain in this appendix. Write it as a "how-to" for setting up your machine:

- **Hardware:** Start with an overview of your environment: what platform(s) did your team develop on (Linux, Mac, etc.)? Give a rough overview of the tech specs (processor,

memory) of the machines. Comment on whether you feel there are any minimum hardware requirements for effective development of your software, beyond "a decent machine".

- **Toolchain:** Next intro and discuss all the software tools you tools used: development environment/editor + plug-ins which were useful/critical, backend databases, other supportive tools/packages (e.g. package managers, etc.) that you installed to make life easier or that are required for other pieces of the software chain to work right. For each one, name it, explain briefly what it does in general, and then why/how it was helpful/needed for this project.
- **Setup:** Now discuss how to actually set up your environment. This should be a step-by-step guide: install this, install that, place this in that directory, etc. You don't have to discuss in detail how to install individual packages (instructions for that are presumably on the site for that package), but do point out any special settings or configurations that should be made in installing it that are relevant to this project. Just imagine yourself walking a new team member through setting up their machine.
- **Production Cycle:** At this point, your instructions should have allowed a newbie to completely set up his/her machine for action. Now let's get down to work: explain the production cycle, i.e., walk through how the edit-compile-deploy process works. It might be helpful here to focus it around a specific example: explain how one might change, for instance, the text that appears in some obvious dialogue in the application...then what you'd do to build and produce a "new version" of your app. To avoid missing small but critical steps, it is useful to just do a little edit yourself and pay attention and write down each step in the process...

In sum, reading this appendix and following the instructions should literally allow a new team member to effectively edit your code and push out a new product version. With that, they are off and running!

Other appendices (optional)

Include any other documents that you feel make your design specification easier to understand but are not central to the project's description.