Computer Science Capstone Design

Assignment: Requirements Specification

Introduction

In real life, a complete requirements document is a major milestone in the development of any software project. The requirements specification is a description of your project's requirements, both functional and non-functional, and forms the contractual basis for the expectations to be fulfilled by the development team. What this means concretely is that the specified features and performance detailed in the requirements document are the basis for "acceptance" of your final software product by the client; if your product meets the stated requirements, the project is closed…and you get paid. From this, it should be obvious that writing a strong and complete requirements document is an absolutely critical skill to master to future software engineers.

As with all deliverables in the Capstone sequence, the underlying aim is to demonstrate and apply the skills you have learned throughout the CS program to a realistic software development challenge. This means that you will need to apply your software engineering training (potentially reviewing previous books/materials, e.g., from Software Engineering) to write up a proper requirements specification. There are also many good resources and examples on the web to get ideas on detailed format and presentation style for your requirements.

Requirements Document Content Outline

As always, the exact details of how to develop and formally present your requirements may vary slightly from team to team depending on the details of the project, but the overall topics and discussion of your requirements specification should include the following:

Cover Page

A nice professional cover page, as usual. Should have the title of the document, the date, the team name, the project sponsor, the team's faculty mentor, and list of team members. Because this document will be revised continually, include some indication of which revision of the document this is (for example, "Version 1" or "Version 2.3").

***For this document**, the cover page should also an acceptance statement:* Just a statement near the bottom of "Accepted as baseline requirements for the project:" followed by side-by-side places to sign and date for "For the client:" and "For the team:" (team lead signs). This will allow you to have a "signed contract" as the semester ends!

Table of Contents

Table of contents should be provided for any document longer than five pages. Just list the major (level 1) headings in the document, with their pages numbers, rather than cluttering things with entries for every tiny subsection.

Introduction (about 1 page)

This section is critical to establish the "big picture" of what your project is about, before you dive into the more focused detail contained in any particular deliverable. Your goal is to demonstrate that you have a deep understanding of the problem proposed by your project sponsor and how the system you will develop addresses this problem.

Always start with the assumption that your reader knows *nothing about your project or its business model* as they pick up your document. This means you have to take the time to explain this overview information; if you do not do this well, your readers will be lost for the rest of the document…if they even read that far.

The rough story you have to tell here goes like this: What is the overall industry/area that we talking about? What are its highlight products/activities? How big is that area/industry (dollars, users, etc.)? You want to convey the size and important of the overall area you are working in.

Next, intro your sponsor and say what he/she does, and how their business fits into the overall business sector you just sketched. Then give some details: How big is their business (volume/users/whatever, how do they make money (or whatever their reward), what is their basic business workflow. Understanding and caring about the whole rest of your discussion is based on getting this big picture across clearly, so this should be one of the best-written sections in the whole document!

Problem Statement (about 1 page)

Now that you have the big picture firmly in place, you have a good basis for explaining what is deficient/broken about the sponsor's business, i.e., what is going wrong…and, ultimately, what you are proposing to build to fix it.

Start by sketching out the sponsor's key business workflow(s); we need to understand how the sponsor's workflow functions before you can tell us what's broken! What is the process by which they produce whatever product/data that is the core of their business? A flowchart or other diagram is often really helpful here to support your narrative.

Then you'll want to describe what the problem is: what are the breakdown, inefficiencies, or missing elements in the sponsor's existing production flow? Start with a sentence or two of overall explanation of the problem, then move to a detailed bulleted list of exact deficiencies or missing capabilities. This is basically the checklist of things you will need to convince us (next section) that your solution fixes!

Note: In your software engineering materials, this description of the broader context may be referred to as "Application Context".

Solution Vision (about 1 page)

Now that we understand the problem well, we can turn to the solution. Begin with an overall statement of what you are proposing to build for the client, i.e., a few general sentences that say what you're building along with the key highlights. Then deepen the detail with a bulleted list of specific features that your solution will provide. It should be evident that the features would solve/address the client's problem!

Next deepen the detail further by going beyond **what** you will provide to sketch some rough detail. Some items you might discuss include:

- What data does your system use and where is this data collected from?
- What data does your system generate and how is this data used?
- What are the main computational processes or data transformations involved?
- How will your system change the way your sponsor works, for better or for worse? What are the trade-offs (for example, does your system impose more work for better results)?
- What other avenues for solving this problem did you consider, and why did you settle on the solution that you did?
- What additional things might your software enable -- in other words, how might your work change the world?

Ideally, your statements about the main features of your system mirror the deficiencies you outlined earlier. Don't be shy about making this connection clear.

Keep in mind that this section should not be overly technically detailed: Your target audience is your customer and others like them, not your computer science professors. That means that you can sprinkle in some technical details in order to convince the knowledgeable and technically-savvy reader that you know your business, but not so much as to overwhelm

A diagram will be invaluable here: This could be an information flow diagram, or the beginnings of a coarse architectural diagram that outlines the big "pieces" of your system, as you see them now. The idea of this diagram is to convey the core pieces of functionality -- the key features of your system, not to be a prescriptive architecture (you'll do that later).

In sum, the Problem Statement and Solution Statement work hand-in-hand to:

1. Introduce a non-technical reader comprehensively to the problem area and your project's place within this space…and hopefully that what you're doing is novel, important, and interesting -- in other words, that they should care about your work and your eventual product.
2. Convince the reader that you have studied the problem carefully to develop a robust, complete understanding of the problem area, and that you have a compelling and complete solution vision in mind to address it.

You're going to reuse this "sales pitch" many times throughout capstone: In this document, in your presentations, during your final capstone presentation, and when you talk about this project with future employers. Do a good job.

Project Requirements (The MEAT!)

This section forms the core of the document and lays out the complete requirements for the system you've just introduced. *You'll want to present these in a "progressive deepening style"*: Begin with a short discussion of overall "domain-level requirements" that lay out the features that the user needs from domain perspective. This then motivates the specific functional, performance and environmental requirements that will be needed to make these overall customer needs happen, described in subsequent sections. For instance, an overall domain-level requirement for a

“secure, responsive, universally-accessible data space accessible to scientists around the world” might drive many more detailed requirements in following sections, e.g., functional requirements for “A user account/profile system”, “secure method of authentication and login”, “cloud-based deployment”, etc.; and performance requirements specifying specific goals for learnability, response time, and usability. You will generally have less than 10 broad domain-level requirements….which motivate and blossom into a large list of more detailed functional/performance requirement. It’s clever if you can indicate the relationship between the two, e.g., numbering the domain level ones, then sub-numbering detailed ones to match.

You will then generally have subsections devoted to detailing:

Functional Requirements (as needed, about 6-8 pages?): Here, you’ll carefully lay out all of the functions that your application will need to have and that you commit to providing. Very often (see your SE materials), these will be hierarchically presented, starting with high-level functions, which are then decomposed into the more detailed, lower level functionality needed to provide them.

Performance (non-functional) requirements (as needed, usually can do in ½ space of functional requirements): Here, your focus is on **how** the listed functions will be expected to perform; this often includes specification of the minimum speed of key computations, accuracy, or responsiveness of the system/interface. It can also include expected usability performance: how fast to learn key functions, training expectations, performance of novice and advanced users, etc. These will commonly be based on assumptions about the end-use hardware/software context, so make any such assumptions you are making explicit!

Furthermore, recall the qualities of *specificity* and *verifiability* that are characteristics of good non-functional requirements statements: Your statements should be specific to your project's needs, not general statements that sound good for all software systems. Your statements should also be clearly *verifiable* and should serve as tests that your system will either pass or fail; for example, I can't test "my system will run very fast" but I can test "my system will retrieve a customer record in 0.5 seconds."

Environmental Requirements (as needed, about 1 to 2 pages)**:** In this section, you will discuss the non-functional requirements related to constraints imposed on your project; these usually come directly from the client, but may also arise from the need to integrate with other software/hardware. In essence, these are things about which you have no choice, e.g., restrictions on hardware platforms, software libraries or products that must be used, programming languages required by the customer, or any other external standards, laws, or constraints of any sort that your system must conform to.

Recall that you want to make these functional specifications as solution-independent as possible: The goal here is to outline *what* your system needs to do and *what* functionality it needs to have, not how this functionality will be built. Something like "the system will allow for administrators to create new user accounts" is both specific and not tied to a specific implementation plan; contrast that with something like "the system will have an "Add User Account" option in the "File..." menu." Even if you may have some ideas about how a piece of functionality will be built, leave those ideas out of it as much as you can: A good functional specification can be implemented in

many different ways by many different teams, and the key to this characteristic is staying decoupled from specific implementation decisions.

Finally, be sure your requirements are clear and well-described. Scenarios and use-cases can be invaluable here, as you work to describe and rationalize the requirements. How you present all of this is a flexible choice: You could have Use Cases as a separate section leading/following requirements presentation, or you could weave them in-line somehow. Same goes for performance specs: they could have their own section, or you could try to weave them in closer to the relevant functions. The overall aim, in any case, is to create a clear, easily-readable, and verifiable set of requirements that will serve as a contract with your client.

Potential Risks (about 1 to 2 pages)

In this section, offer your analysis of the risks that are most relevant to your project as well as the *impacts* of these risks. What determines relevance? This could be based on the likelihood of the risk occurring, or the effect of the risk on the overall success of your development effort.

Recall that the risks documented here aren't those that of relevance to you (for example, "there is a risk we fail to learn some new technology in time") but those of relevance to your system and your customer. For example, if your system gets a calculation wrong, what is the risk this mistake could cause? Does someone just get the wrong answer? What happens because of this mistake? Is someone merely inconvenienced or, worst case, does someone die? What happens if a competitor brings out a new product? Is your project dead, or are there ways to adapt? Think of how your software could affect the user and those that rely on your user's actions.

Project Plan (about 1 page)

Offer a short discussion of your project execution plan, as it stands right now. Describe a number of milestones (as you begin, 5-10 milestones should be easily identified, but this will become more finely granular as you continue working), in terms of the functional requirements for the system (or groups of functional requirements), and lay out when these milestones will take place in the months to come. A graphical depiction like a Gantt chart should be included, discussed and supported by your narrative text.

Conclusion (about ½-1 page)

Every document must have a conclusion. Remember: Really busy executives, reviewers, and division managers often read the intro and conclusion sections in detail…and just scan the rest. And even readers who've slogged through the whole thing need you to bring it all together for them. Your goal in any conclusion are the same:

- Remind us of the important of the problem and the project
- Review what the problem is, and sketch the solution you have in mind
- Review what you did in this document that contributed towards project progress
- Summarize any key insights, and make a positive statement of your progress and foreseen outcomes.

Glossaries and Appendices (as needed)

If you use any terms that have special meaning (domain-specific terminology, for example), lay out the definitions here. Similarly, attach in appendices any ancillary documents that you feel make your requirements specification easier to understand or provide additional detail, but are not central to the project's description.

DELIVERABLES

A complete (!!) draft of the requirements document, professionally presented to your Team Mentor. Since this is a draft, you don't need to bind it; staple is okay.

A final hardcopy document, professionally presented. This document will be delivered to your Team Mentor at or before the time of the final exam, and must be signed off and dated by your client and your team lead on the cover page.

Important Notes:

- You will, of course, want to review the evolving requirements document continuously with your client leading up to December, to discuss and agree on specific points on this contract between you. If the first time your client sees this document is when you present the final pretty version to sign off on shortly before finals week, you have made a serious mistake!
- The draft document is the first deliverable, and will be graded with *exactly the similar completeness/quality expectations as the final document*, but for only 20pts. This gives you a chance to *learn* and to *improve* your written deliverable before the final version is graded. The final deliverable will be re-graded using the same expectations, but for 100 pts; this makes the total weight of the Requirements Assignment 120 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.