Writing a Project Proposal

An important skill for Computer Science professionals is the ability to simplify technical concepts for unfamiliar audiences, and to explain technical decisions. Often, this comes in the form of writing “design docs” which describe how a system is designed and are intended to be read by other engineers, often those who might be joining the team that maintains a system. Describing a whole system–in this case, your project–may seem daunting at first, but all design docs generally have the same core parts: a brief overview of functionality (for you, this is your project pitch), a description of functionality, and a list of components in the project. Design docs for new systems (i.e. your project) also typically include a schedule, a list of possible risks, and a method for dividing work. Below, we’ve provided guidance on how to write each of these sections.

When you’re writing a technical document, you should always keep your audience (who will be reading the document) in mind: as you write each section of a design doc, ask yourself if it would make sense to your audience, and if your audience might have any questions about the section–if you think of some questions, answer them in that section so your audience doesn’t have to remember to ask you later. For your CS222 project proposal, assume your audience is technical, but may not be familiar with your domain of Computer Science. For example, if you are building a website, you should assume your audience knows how to code, but may not be familiar with specific web development terminology. When in doubt, it’s better to have too much detail than too little, but make sure not to over explain concepts as if your audience is not technical.

Pitch

A pitch sets the tone for the rest of a proposal, and is the first impression that stakeholders, from investors to managers, will have, so provide an overview of your project’s functionality and benefit to users (and, if it solves a specific problem, mention that too). Avoid getting bogged down in technical details, such as what programming language the project uses. A good pitch is clear and concise, and 1-2 sentences long.

Functionality

Include a list of 5-10 specific actions that users can complete in your project. Each action should be specific, such as “Users can view a list of their photos,” but do not mention *how* the action will be performed (do not say “Users can tap the home button to see a list of their pictures”) since this might change as you develop your project. Each of your 5-10 actions should stand alone, meaning they do not reference each other, and completing or not completing one item does not impact your ability to complete another (do not say “Users can use the photos list to share pictures through email,” say “Users can share pictures through email”).

As you're completing this list, make sure to consider how much time you are going to commit so that the scope of your project is achievable.

Components

Next, describe each major part (i.e. large classes or groups of classes, not individual methods) of your project in detail, and include the following for each part:

- *Functionality.* State the role of the component in the system, and a list of tasks that it is responsible for. Explain why you chose to divide this set of features into its own component.
- *Programming Language(s) used*. You should explain why you are making this choice given the unique characteristics of different programming languages and your team's skill set. If you use the same programming language for every component, you only need to include one explanation, but it should address why that programming language is used for the entire project.
- *Major Libraries Used*. If you know your component will heavily rely on an open source library, you should specify what library that is and why it is a good choice to use.
- *Testing Methodology.* Provide a brief overview of how you plan to write programmatic tests to ensure that this component behaves as expected. Explain why this testing strategy is appropriate for your application. If the component is something you cannot test, explain why.
- *Interactions With Other Components.* Specify what other components, if any, in the system this component will communicate with, and how that communication will occur (such as a method call, an HTTP request, etc). Explain why you chose this method of communication.

You may organize this section however you feel is most readable, and the above is only a guide for organization. For example, you could explain your division of components after explaining each component in detail.

Once you've described the components, include a diagram showing how they communicate with one another (as a general rule of thumb, if Component A sends messages to Component B, you should have an arrow from Component A to Component B).

Schedule

After all of the proposal revision is complete, you'll have eight to ten weeks to work on your project. For each of these eight weeks, describe at least two specific high-level tasks you plan to complete. Each task should be functionality focused and not mention specific small methods you plan to implement – that's too hard to predict. A good task might be "Create user interface elements to build login page with link to Google," since it is specific about what the end result will be (a login page with a link to Google), but is also high-level and does not specify how the user interface elements will be created or what they will be.

Scheduling is extremely difficult, and we will not require that you perfectly stick to your schedule: budgeting time for tasks is a skill that only improves with experience. Instead, make sure that the schedule is evenly balanced across weeks and each week has at least two tasks.

Risks

In the context of a technical project, a risk is just something that can go wrong and impact your timeline or your bottom line. You should include at least three possible risks, as a plan for what happens if each risk materializes. This plan should include the steps you will take to resolve the issue, the expected impact of this to your schedule, and how you will adjust your schedule as a result. It’s perfectly okay if your method of resolution is to ask your mentor for help, although this shouldn’t be the method of resolution for every risk.

Teamwork

Having a consistent development environment reduces friction on your team and eliminates the problem of code working one one person’s computer but not on another person’s computer. Although it may be a bit of work to set up, there is payoff in the long run.

Describe at least one way you plan to reduce friction when collaborating on your team, and explain why it makes sense given who is on your team and what their development environments are.

Then, describe how you plan to divide up the work for the project. This can be any way you want, but you should make sure that it results in a fair division of work. Be careful to not assign specific small tasks, such as writing an individual method, to team members; instead you should assign specific pieces of the project, for example “Ryan will create the login page.” Or, if you have strong cohesion with your team, you could make no predetermined assignments and have everything be on a volunteer basis. Ultimately, this is up to you, and anything fair and reasonable goes.