How to Write a Design Document

Overview

- Be sure to read through this entire page. It's all relevent.
- Each design document is worth 33.33% of the project. While it will likely take less than 33.33% of the time you spend on the project, you should take it very seriously.
- We will grade your designs harshly. The design is essentially the most important part of the project. Having a good project design can literally cut your total coding time by a factor of 10.
- Design documents should be around 2,000 to 4,000 words long. If it's longer than 5,000 words, we won't read it. So keep them short and to the point.
- Design documents will be submitted to the project server (`klwin00.ucmerced.edu`) in PDF format. We encourage you to use your favorite word processor to make your design document, but you must convert it to PDF when you're done.

What goes into a design document?

A design document is a complete high-level solution to the problem presented. It should be detailed enough that somebody who already understands the problem could go out and code the project without having to make any significant decisions. Further, if this somebody happens to be an experienced coder, they should be able to use the design document to code the solution in a few hours (not necessarily including debugging).

So, what actually goes in?

- A high-level description of your solution, including **design decisions** and **data structures**
- **Declarations** for all new classes, procedures, and global/class variables
- **Descriptions** of all new procedures (unless you can tell exactly what it does from the name), including the purpose of the procedure, and an explanation of how it works and/or pseudocode

For example, this is the pseudocode one would write for the existing `Condition::Wait`:

```
void Condition::Wait()
   waiter = new Semaphore, initially 0
   append waiter to waitQueue
   conditionLock->Release()
   waiter->P()
   conditionLock->Acquire()
```

Your pseudocode has to be precise. For instance, in describing your solution for the `Communicator` class, it is not enough to say

```
 if anybody's waiting, then signal cv
```

You need to say how you determine if anybody's waiting, e.g. by saying

```
 if (waitingReceivers>0 or waitingSenders>0) then signal cv
```

Also, another important thing to remember is that a design document needs to include the **correctness invariants** and **testing strategy**. The testing strategy includes a clear plan of the testing methodology and may include a description of test cases that will be used to test correctness invariants. Focus on the testing strategy. If you want, you may itemize things that will need testing.

What *doesn't* go into a design document?

Keep in mind that your TA understands the project very well. Do not restate the problem in your design document. Your TA is far more interested in your solution than in knowing that you understood the problem.

Your design document should contain very little actual code, if any at all. Include psuedocode for all complex procedures, but do not include Java code.

The purpose of pseudocode is to avoid all the annoying aspects of programming languages that make code both harder to write and harder to read. The purpose of pseudocode is *not* to be imprecise about how you solve a problem. Comments are welcome, but ASSERT and DEBUG statements, for example, do not belong in pseudocode.

Remember that we have to *read* your design documents. If you don't think we want to see it, don't put it in!

Be sure to make sure your .pdf file works before you turn it in (try viewing it with gs, gsview, Preview, acroread, etc.).