

Writing a Design Document for Computer Science 51

CS 51 Course Staff

February 17, 2007

1 Introduction

A major purpose of CS 51 is to get you to think carefully about designing software. As such, writing thoughtful design documents is an integral part of the course.

2 Why Design

Writing design documents develops and hones your ability to think through a design and reduce the number of costly and unplanned changes to your software program. In an ideal world, you would maintain and continually update a design document as a programming project proceeds (e.g., for the purposes of documentation and training), but we won't go quite this far. The design document is designed to get you to think about the details of each project in advance and to practice communicating succinctly about what you are doing.

3 Beware!

Design documents are due far before the problem set is due. Take advantage of this: start thinking about how to design your program before you start writing it.

Think first. Code second. Be done early and enjoy life.

4 Contents of a Design Document

We expect you to be mature programmers having experience from CS 50 or prior coursework. We do, however, still mandate a strict design document format. You must include each of the sections and must label each section.

This strict organization is necessary because design documents may have multiple audiences. Careful organization into subcategories allows each reader to focus on the relevant portion of the document. The testing team is not interested

in the abstract, and the corporate VP is not interested in the architecture. The goal is not to write a carefully crafted essay with smooth paragraph transitions, but a focused, organized document.

Problem statement. In one or two sentences, state the purpose or problem that your program will solve. Make this a short, high-level overview. What would you tell your roommate about the project?

Behavior and Assumptions. In a few sentences, please describe how your program will be used. Include in this a description of the actors that will interact with your program. E.g., will your program be used as part of a larger system or used directly by a human? Most importantly, give an example of your program in use—for instance, show the inputs and the output for a single run through the program to give the reader a better sense of what the program will do. Also list any assumptions that you are making about the types of input you will be able to handle—where does input come from, what happens after the program executes, are there any additional constraints that must be followed by your solution. This section is more important the more open-ended a problem is.

Architectural design. Describe the major components (functions and data structures) of your solution and illustrate how these components interact. Feel free to draw diagrams. Focus on the novel components of your design, i.e., don't spend time explaining the function of common libraries or components built in previous assignments.

Your design should focus strongly on the interfaces between components of your program. What contracts will each function implement?

This is the most important section of your design document, and will take you the most time to write. It will not only require a full understanding of the assignment, but also will require serious thinking about how you plan to attack the problem. Pseudocode is not strictly necessary, but often helps people do a good job on this section.

Implementation strategy. In what order will you put together your program? Why? How long do you expect each component to take you? What schedule will you give yourself to complete the project—include specific dates!¹

The major key for this section is to identify ever-increasing subsets of the implementation that are runnable and fairly quick to build. You should strive to maintain a working, well-tested program from the very beginning and add features and requirements on top of it. How can you best do this?

Remember, programming is easier and more fun when you're continually working from a code base that actually does something.

Test plans. How will you test your partially working program? What kind of testing framework will you use to demonstrate that your final program works? What test cases will you use? Be specific. Give examples the test cases you plan to use—you should include corner cases and error conditions.

¹While this may seem like a strange exercise, in the real world you will often be asked to estimate how much time your work will take. Practicing this skill now will make your answers more credible when it counts.

4.1 Programming and debugging

Once you've finished your pre-coding write-up, go ahead and start with item 1 of your implementation plan. As you proceed with coding and debugging, you'll probably discover use cases, requirements, and corner cases that you did not anticipate. You may even completely change your architectural design. That's all fine. While we'd like you to think about your design decisions, you won't necessarily have to update your design document to reflect all of your new decisions. We'll let you know on a per problem-set basis.

5 Finding Help

Design is hard. Don't be discouraged if you find that you don't know how to get started with the first several programming assignments. The writing of the design document is a low-cost way of finding that you're actually quite confused. Take what you have written and go see a TF or the professor. We love to help, and we know what questions to ask to get you on the right track.

A Frequently Asked Questions

1. *Do I have to write a design document?*

Yes.

2. *What's the point of writing a design document?*

We want you to learn how to design programs—not merely to think before you code but because design is one the fundamentally interesting aspects of computer science.

3. *How long and how much detail should be in my design document?*

Great question. Spend enough time to focus on the novel features of your architecture, but not to the point of providing detailed code—this should be a bigger-picture overview showing how your program's components interact. You should have a detailed implementation plan—not just, “I will do this in the order outlined above”, and you should provide multiple specific test cases covering different possible classes of input.

Overall, we're looking for a clear, concise, and simple specification for your proposed implementation.

B Example

This appendix presents an example of what we expect in a CS 51 design document based on implementing a function to sort a list of numbers by an arbitrary comparison function.

B.1 Problem

We would like to implement a sorting algorithm that will be used by other programmers for their own sorting needs.

B.2 Solution

B.2.1 Problem Statement

Write a function that takes as input a list of numbers and a comparison function and sorts that list.

B.2.2 Behavior and Assumptions

This function could be used almost anywhere—for instance, a programmer may wish to sort temperature readings over the summer. Such a use might look as follows:

```
// this example will sort the input in ascending order

int comparison( float a, float b )
{
    if ( a < b )
    {
        return -1;
    }
    else if ( a == b )
    {
        return 0;
    }
    return 1;
}

// a float is just a number that may contain a fractional component
float my_list[] = {0.3, 9.4, 5.0, 89.9, -1.2};
sort( my_list, comparator );
```

The sort function will have two arguments, an array of floating point numbers as well as a comparison function that provides an ordering over the floating point numbers.

- the input list will be an array and should not be NULL

- The list must consist of floating point numbers
- The comparison function will be a function pointer and should not be NULL.
- The result of the sort will be to change the value of the input list.
- The validity of the inputs will be checked via assertions
- The list itself will be sorted in place, destructively
- The comparison function should provide a total ordering
- The comparison function will take two inputs, a and b , and return -1 if the $a < b$, 0 if $a = b$, and 1 if the $a > b$.
- The sort function will sort in ascending order as defined by the comparison function
- The sort algorithm must be efficient and reliable— $O(n \log(n))$ in the worst case
- The sort may use additional memory beyond the input list

B.2.3 Architectural Design

The sorting function will be implemented with a merge sort algorithm to provide for $O(n \log(n))$ worst case performance.

Merge sort can be broken into two distinct phases:

- Split the list into two halves and recursively sort them
- Merge the sorted lists together

This specification maps directly to the design. A recursive merge algorithm will take a list, split it in a half, sort each half, and then merge them together. The base case is simply when there is either one or zero elements in a list; nothing changes, as it is already sorted.

We will have one helper function, `merge`, with the following prototype:

```
bool merge( const double *first, int first_size, const double *second,
int second_size, double *result );
```

The interface is as follows

Inputs

- `first` is a sorted list of numbers; it will not be changed inside merge
- `first_size` is the length of first
- `second` is a sorted list; it will not be changed inside merge

- `second_size` is the length of `second`
- `result` is uninitialized (but properly allocated) memory of size `first_size + second_size`

Either `first` or `second` may be `NULL`, in which case the corresponding size must be 0. In this case, the contents of the other list is copied into `result`. If both inputs are `NULL`, then `result` is unchanged.

Notice that we used `const` as part of the function prototype. This has two functions: first, it demonstrates that the contract of the function is such that it will not modify values stored in either of the two arrays. Second, it provides language enforcement for this contract by explicitly disallowing the programmer from changing the values stored in either of the two arrays.

Outputs On error, `merge` returns `false`. Errors could include `result` being `NULL` or `first` or `second` not being in sorted order (although it is not mandated that the implementation check to ensure the lists are in sorted order).

Second, the array `result` stores the merged lists and functions as a second output of the function.

B.2.4 Implementation Plan

The first step in the implementation is to provide a simple framework for testing. A main loop will be written that reads in a set of numbers from the user as well as a list sorted in ascending order, stores the unsorted input in an array which will be passed into the merge sort algorithm, and then compares the result of the sort with the sorted input list.²

The second step in the implementation will be to write the core of the algorithm: the `merge` function. This will allow us to then build the recursive sort around this working code.

Clearly, testing the `merge` function first will allow us to proceed with more confidence in implementing the recursive merge sort.

The entire project should take under 5 hours all in a single day, on, say, Feb. 8, 2006. Two hours will be budgeted to implementing and testing the merge algorithm. These tests will be discussed in more detail below.

B.2.5 Test cases

Testing the sorting algorithm will involve checking several specific cases, both valid and invalid.

- A single element list (e.g., the list containing only the number 1)
- A two element list, one sorted correctly and one sorted in reverse order—e.g., 1,3,2 and 2,1,3

²This is not discussed in the architecture section because it is merely a component of the test framework, but your actual implementation plan should include code designed to help you test your program.

- Sorted and unsorted lists of a size that is not a power of two—e.g., 1.3, 2, 4.5 and 2, 1.3, 4.5
- A list with no elements
- A list with negative numbers: $-1, 2, 3, -2$
- A list with multiple copies of the same element: 1, 1, $-3, 3, 3, 2, 2$

(As an aside, it might also be worthwhile to time the execution of the code to ensure that the algorithm is actually running reasonably fast.)