**CS 312: Overview Documentation for Programming Assignments**

Every programming assignment that you submit should be accompanied by an overview document, which will be graded. An example programming assignment, including an overview document, is available on the course web site for your consideration.

**Format.** You should format an overview document as either ASCII text (with lines at most 80 characters long), HTML, Microsoft Word (**.doc**), or PDF. The filename must be **overview.\***, where \* is the appropriate extension. Use good taste in formatting your document; the course staff values simplicity and readability over fancy graphic design. The typical length of an overview document should be two to four letter-sized pages, with 10–12 point font and reasonable margins. Use good judgment about what will be useful to the course staff while grading your assignment: Incomplete and excessive documentation will both be penalized.

An overview document should be structured as described below. The suggested word length of each section is a guideline, not a requirement.

**Metadata**

* The programming assignment title—e.g., "Problem Set 1: SML Warm-Up."
* The names and netids of all students in the group submitting the assignment.

**Summary (100-300 words)**

**Summarize the most noteworthy aspects of the rest of your overview.** Anything you mention here should be described in more detail later in the document. You might include:

* Which parts of the assignment were the most challenging.
* Interesting design decisions that you made.
* Major issues that you ran into while designing, implementing, or testing your program.
* Known problems with your implementation, such as missing functionality or bugs.

**Specification (10-500 words)**

Some assignments will provide complete specifications. Other assignments will be incompletely specified, and it will be up to you to finish the specification. You may, for example, need to fix inconsistencies, fill in omissions, or resolve ambiguities.

**Describe your finished specification.** Identify the choices you made in finishing the specification, and provide a rationale for them. It might be useful to discuss alternatives you considered, why you rejected them, and the tradeoffs involved in your ultimate choice. If you made any changes to the given specification, justify why a change was needed and why the result of your change is reasonable. We encourage including (brief) actual SML signatures and comments that document them.

Your job is to convince the course staff that your finished specification is reasonable and in the spirit of the assignment. If you are not convincing, the staff may deduct points.

**Design and Implementation (150-1000 words)**

Discuss how you created a program that satisfies the assignment specification. This includes topics such as:

* Modules:
  + How you decomposed the program into modules, and what the purpose of each module is.
  + Key invariants maintained by modules.
  + Inputs and outputs of modules.
* Architecture:
  + How modules depend on each other, and how modules are connected (e.g., layers, pipes, filters, ...). Diagrams may be helpful in describing your architecture.
* Code design:
  + What algorithms are employed in your program. We encourage including pseudocode to document important algorithms. We also encourage discussion of any boundary conditions or formulas involved, as well as citations for sources of your algorithms.
  + What data structures are employed.
  + Tradeoffs you made between simplicity of code and efficiency of execution.
* Programming:
  + The implementation strategy you used—e.g., top-down, bottom-up, ...?
  + Any challenges you encountered while coding.
  + Any code reuse or revision you employed.
  + Which group members wrote which parts of the code.

Your job is to give the course staff enough information that we can understand your program, without needing to look at the source code, and to convince us that you thought carefully about the construction of your program.

**Testing (150-1000 words)**

The course staff will often provide some test cases for you; it is almost always a mistake to think that those test cases are complete. The course staff often has a reserve of tricky test cases that your code will be tested against. So spend time designing your own test cases. Don't wait until the last minute—do it early in the assignment, even before you've written much code.

**Describe and evaluate your test plan**, and discuss any interesting issues that arose during the design of your test plan. Why do you have confidence that your test cases have enough coverage of the input space, and enough coverage of your code? Do your test cases accomplish both unit testing and integration testing? A short summary of your actual test cases may be helpful. For some assignments, you may also be asked to attach actual test cases and results as a separate document.

**Describe the results of carrying out your test plan on your program.** Which of your test cases pass, and which fail? Remember, testing should not be something you do after finishing coding; rather, you should be testing your program incrementally as it is developed. A few examples of test cases and your program outputs may be helpful to illustrate the functionality of your program. But don't overdo this: The course staff is not interested in seeing reams of test case results.

Claimed test case results that cannot be reproduced with your submitted code will be severely penalized and might even be an academic integrity violation. Never "fix up" the output of your program.

Your job is to convince the course staff that you exercised due diligence in validating your program.

**Known problems (1-500 words)**

Detail any known problems with your specification, design, or implementation. Problems that are clearly identified and described here will not be penalized as much as problems that the course staff discovers on its own. Are there bugs in your program? Missing functionality? Incorrect specifications? Incorrect documentation in the code? Let us know.

**Comments (1-200 words)**

**Express your opinions about the assignment.**This section of the overview document will not be graded, but it can earn good karma. You might address such questions as:

* How much time did you spend on the assignment? How was it divided between designing, coding, and testing?
* What advice should we have given you before you started?
* What was surprising about the assignment?
* What was hard about the assignment?
* What did you like or dislike about the assignment?
* What would you change about the assignment?