CSci 5801: Software Engineering I, Fall 2019
Project 1 – Waterfall Methodology
Task 3: Code, Documentation, Testing, and BugList
Due Date: Monday, November 18th, 11:55pm

Special Instructions: You will be working in your small groups to complete this project assignment. You should meet, skype, or talk on the phone (if unable to meet in person) about the requirements for the assignment. You will only turn in one assignment per group. You must include all names on your assignment with X500 names and your Team # on all documents. Please use the name that is listed on the class roster so we will know who you are. You will upload your work to your team repository on GitHub.

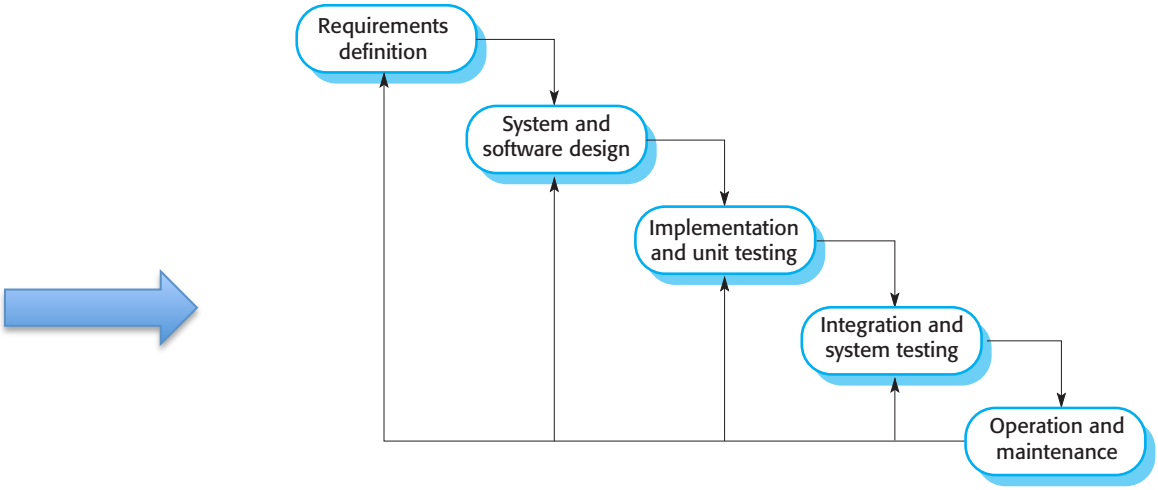
The Problem

There are numerous types of voting algorithms and in the United States, we typically use plurality voting where each voter is allowed to vote for only one candidate, and the candidate who polls the most votes is elected. It is rare for an election to be tied but if that occurs, there is typically a runoff between the tied candidates. For example, there have been three cases in history where there was a tie in the Electoral College for a presidential election. The House of Representatives then decided who was president by voting. For small sized, local elections a run-off may occur or even a coin flip can decide the outcome in some cases. Much research has been performed on voting theory and some believe that the other types of voting are better than our style of voting.

You are tasked with creating a voting system that is capable of performing Open Party Listing (OPL) voting and Closed Party Listing (CPL) voting. The input file will indicate what voting algorithm should be used (i.e. OPL versus CPL.)

Your Work for This Piece of the Project

You and your team have been assigned to the task of developing this voting system and you will be using the Waterfall methodology.



Your Project Task

You and your team have finished the first two stages of the Waterfall process for the system. Now, you and your team will code your project using your software requirements specification (SRS) document and your software design document (SDD) to guide you. You will complete your own unit testing and provide documentation of these unit tests as part of the deliverables. In addition, you will run tests to ensure the program as a complete system is working and that

the software requirements have been fully meet (i.e. integration and system testing). Since the operation and maintenance phase is the phase that occurs only after the system has been put into practical use, we will only document the issues and bugs that have not been fully fixed that be would addressed as part of the operation and maintenance phase. This bug list will be used as a starting point for Project 2. Project 2 will use an iterative processes called Agile Scrum. Some teams will have fully working code and some teams may not. The expectation is that you will document the state of your code.

Asking Questions

You can bring your questions to class, and I will try and answer them at the end of class.

Deliverables

1) We will be using GitHub for this portion of the Waterfall process. All work will be uploaded to your team's GitHub account. We will expect all documents, code, tests, and bug list to be in your team's repository and directory structure as stipulated below. We will clone your team's work to a CSE machine to compile, run and verify your code.

GitHub Team Directory Structure:

umn-csci-5801-f19/repo-TeamXXX	: XXX is your team number, all teams have a repository set up
/Project1	: Create directory in your team repository to store all work
/src	: Create directory under Project1 to store all your program files : Be sure to include your makefile if using C++ : We will compile your code so only provide the program files with the : code
/testing	: Put all test logs along with all test files that were used for testing here (e.g. : CSV files used for testing)
/documentation	: Place all documentation documents here. We expect you to use javadocs : or doxygen to generate your formal documentation of the code.
/misc	: If you have other files that you are unsure where to store, place them here. : Use good naming conventions for all files
Readme.md	: This is stored in the Project1 directory and should provide instructions for : us if there is any special handling or issues we should know about.
buglist	: This is stored under the Project1 directory. You will use the provided : template to document your bugs and issues. See buglist description below : for template information.

2) All source program files will be provided and stored in the proper directory.

- Correctness and Program Execution: Your code should execute with no syntax or runtime errors and produce correct output. Your program must meet the specifications and function properly.
- Readability: Your code needs to be easy to read and follow. It should be stylistically well design.
  - Use indentation consistently.
  - Variables, methods/functions, and files should have meaningful names. Please do not try to be funny or vague. We are expecting professional looking code in a 5000 level course.
  - Code is well organized. Methods and functions organized into blocks of code that can be reused. Do not, I stress, do not, create huge methods or functions. A good rule to follow is that methods and functions should do only one thing and do it well. Sometimes it makes sense to have a method or function to a couple of things but this grouping must be for a reason. You are expected to pass variables and use method/function calls.
- Documentation:

    - Every file should have a header comment. It should contain the name of the file, the description of what the code does, and the name of the author.
    - Commenting of the code itself is expected. You need to explain what is happening in the code itself.
    - Use either javadocs or doxygen to create the formal documentation of the code itself. All methods/functions must have comments that will generate documentation via javadocs or doxygen. You cannot have a method/function without its purpose being documented.
- Efficiency: Your code should be efficient in its processing. For example, your program may take too long to run or your have blocks of code that could be written more efficiently by reducing the number of lines of code. Example: use looping constructs when needed instead of copying and pasting code over and over.
- Assignment Specifications: Please ensure you provide the files in their proper locations as defined in the GitHub section of the deliverables.

3) Documentation (beyond the comments in the code itself as part of documenting flow):

- You will use either javadocs or doxygen to generate the formal documentation that would be provided to a user or programmer. You should ensure that you document each class, method/function, header file, etc with the name, description of purpose, input parameters (with purpose), return value (with purpose), and exception handling. We should be able to read the documentation and understand exactly what your class, headers, and methods/functions are doing.
- Store all files generated for your documentation under the /Project1/documentation directory on GitHub.

4) Unit Testing: You will be testing program components such as methods or object classes to ensure the functionality of these components work properly. You will create logs for each of the units test and run the unit tests to determine whether each test passes or fails. All methods and object classes will need to be tested. You can use a testing framework such as JUnit (for Java), Google test framework (for C++), or write your own methods/functions to run your tests. We will expect your tests to be logged and your code for testing provided to us so we can run your tests also. We will be covering testing during Week 8 of the class. You should read Chapter 8 of the book focusing on Section 8.1 (Development Testing). Use the provided test case log template to document your tests – on Moodle.

- Each unit test will require a test case log. You should have one file with all test case logs (both unit and system tests). Name your test case log file, *testinglogs.XXX* where XXX is the file extension (e.g. pdf, docx).
- You will put your log file in the /Project1/testing directory under your team repository. Your code for the tests will be included in the /Project1/src directory.
- All CSV files used for any testing should be placed in the the /Project1/testing directory. We will move files around as needed when testing your code on a CSE machine.

5) System Testing: You will test to ensure the entire program works as the specifications required. You will create a log file for the test case and document the inputs and files that you used while testing.

- Each system test will require a test case log. You should have one file with all test case logs (both unit and system tests.) Name your test case log file, *testinglogs.XXX* where XXX is the file extension (e.g. pdf, docx).
- You will put your log file in the /Project1/testing directory under your team repository. Your code for the tests will be included in the /Project1/src directory.
- All CSV files used for any testing should be placed in the /Project1/testing directory. We will move files around as needed when testing your code on a CSE machine.

6) BugList: You will create a log of bugs. The log should be a table that looks like this:

	Description of Bug (be specific About what is not working)	Location of Bug (file, class, method, etc)	Steps to Recreate Bug and/or Test Case #	Root Cause Analysis Notes

- Name the file buglist.XXX where XXX is the file extension (e.g. pdf, doc, docx).
- Place the file under the Project1/ directory
- Remember, it is better for you to catch your own bugs instead of the grader. You will be docked points if something does not work but if it is on the bug list you will not have full points taken away for that part of the code.

Due Dates

All deliverables are due on: Monday, November 18th at 11:55 p.m.